195 PTAS.
(IVA Incluido)

117

mi computer

CURSO PRACTICO DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR

P.V.P. Canarias, Ceuta y Melilla 185 Ptas.



MI COMPUTER CURSO PRACTICO
DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR

Publicado por Editorial Delta, S.A., Barcelona

Volumen X-Fascículo 117

Director: José Mas Godayol
Director editorial: Gerardo Romero
Jefe de redacción: Pablo Parra
Coordinación editorial: Jaime Mardones, Francisco Martín
Asesor técnico: Ramón Cervelló

Redactores y colaboradores: G. Jefferson, R. Ford, F. Martín, S. Tarditti, A. Cuevas, F. Blasco
Para la edición inglesa: R. Pawson (editor), D. Tebbutt (consultant editor), C. Cooper (executive editor), D. Whelan (art editor), Bunch Partworks Ltd. (proyecto y realización)

Realización gráfica: Luis F. Balaguer

Redacción y administración:
Aribau, 185, 1.°, 08021 Barcelona
Tel. (93) 209 80 22 - Télex: 93392 EPPA

MI COMPUTER, *Curso práctico del ordenador personal, el micro y el miniordenador*, se publica en forma de 120 fascículos de aparición semanal, encuadernables en diez volúmenes. Cada fascículo consta de 20 páginas interiores y sus correspondientes cubiertas. Con el fascículo que completa cada uno de los volúmenes, se ponen a la venta las tapas para su encuadernación.
El editor se reserva el derecho de modificar el precio de venta del fascículo en el transcurso de la obra, si las circunstancias del mercado así lo exigieran.

ISBN: 84-85822-83-8 (fascículo) 84-7598-183-6 (tomo 10)
84-85822-82-X (obra completa)
Depósito Legal: B. 52-84

Fotocomposición: Tecfa, S.A., Pedro IV, 160, Barcelona-5
Impresión: Cayfosa, Santa Perpetua de Mogoda (Barcelona) 158604
Impreso en España-Printed in Spain- Abril 1986

Editorial Delta, S.A., garantiza la publicación de todos los fascículos que componen esta obra.

Distribuye para España: Marco Ibérica, Distribución de Ediciones, S.A., Carretera de Irún, km 13,350. Variante de Fuencarral, 28034 Madrid.
Distribuye para Colombia: Distribuidoras Unidas, Ltda., Transversal 93, n.° 52-03, Bogotá D.E.
Distribuye para México: Distribuidora Intermex, S.A., Lucio blanco, n.° 435, Col. San Juan Tlihuaca, Azcapotzalco, 02400, México D.F.
Distribuye para Venezuela: Distribuidora Continental, S.A., Edificio Bloque Dearmas, final Avda. San Martín con final Avda. La Paz, Caracas 1010.

Pida a su proveedor habitual que le reserve un ejemplar de MI COMPUTER. Comprando su fascículo todas las semanas y en el mismo quiosco o librería, Vd. conseguirá un servicio más rápido, pues nos permite realizar la distribución a los puntos de venta con la mayor precisión.

Servicio de suscripciones y atrasados (sólo para España)

Las condiciones de suscripción a la obra completa (120 fascículos más las tapas, guardas y transferibles para la confección de los 10 volúmenes) son las siguientes:

a) Un pago único anticipado de 27 105 ptas. o bien 10 pagos trimestrales anticipados y consecutivos de 2 711 ptas. (sin gastos de envío).

b) Los pagos pueden hacerse efectivos mediante ingreso en la cuenta 6.850.277 de la Caja Postal de Ahorros y remitiendo a continuación el resguardo o su fotocopia a Editorial Delta, S. A. (Aribau, 185, 1.°, 08021 Barcelona), o también con talón bancario remitido a la misma dirección.

c) Se realizará un envío cada 12 semanas, compuesto de 12 fascículos y las tapas para encuadernarlos.

Los fascículos atrasados pueden adquirirse en el quiosco o librería habitual. También pueden recibirse por correo, con incremento del coste de envío, haciendo llegar su importe a Editorial Delta, S.A., en la forma establecida en el apartado b).

No se efectúan envíos contra reembolso.

Editorial Delta, S.A.



Ingeniería

Ahora nos referiremos a la ingeniería en hardware y en software, carreras relacionadas con la instalación y mantenimiento de nuevos sistemas

El concepto de ingeniero de hardware, la persona que diseña y construye sistemas de ordenador, aparece rodeado de un aura mítica; no obstante, se trata de una actividad de características perfectamente delimitadas y definibles.

La etapa inicial es la de la especificación: decidir cuáles son las funciones que ha de llevar a cabo la nueva máquina. Esta etapa tiende a ser competencia del ingeniero más experimentado, en unión con el departamento de marketing. En el campo militar y de defensa (en Gran Bretaña, p. ej., hay más ingenieros de hardware trabajando en proyectos de carácter militar que en proyectos civiles), la especificación es privativa del Ministerio de Defensa.

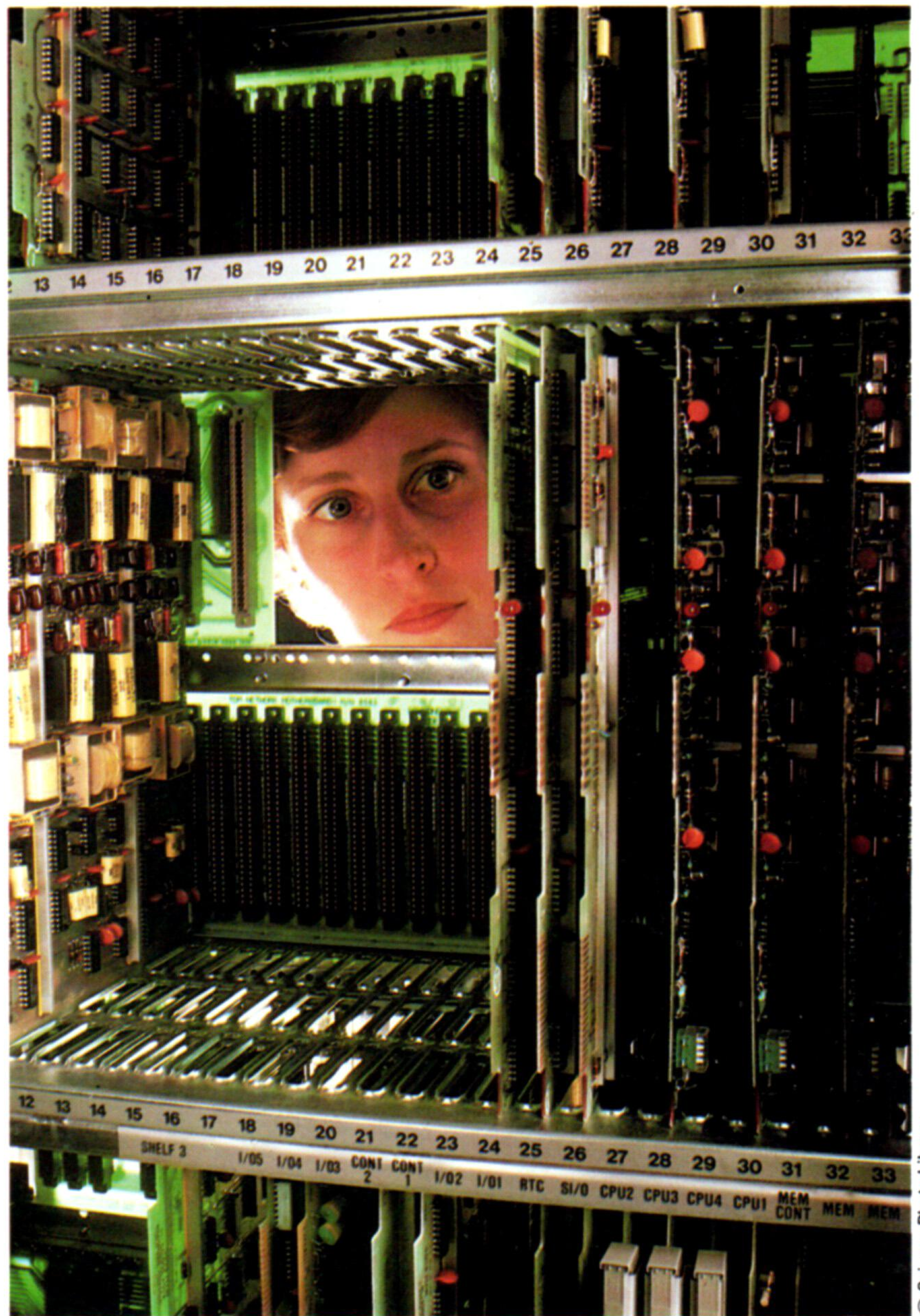
La siguiente etapa para el ingeniero, tras recibir las especificaciones, consiste en decidir la viabilidad del proyecto y cuánto tiempo llevaría su realización. (Los plazos representan la esencia de todos los proyectos de ingeniería, tanto militares como civiles.) Habiendo calculado el costo del proyecto y evaluado las limitaciones de tiempo, el ingeniero debe diseñar el sistema. Para adoptar una decisión acerca de los componentes a utilizar y el modo de conectarlos, los ingenieros han de estar al tanto de los últimos avances tecnológicos a nivel de componentes y, en consecuencia, habrán de ser asiduos lectores de la prensa especializada. Tras el diseño inicial, habrán de ensamblarse y probarse varios prototipos antes de que se pueda comenzar la producción final.

Diseño de prototipos

Toda ingeniería inevitablemente es un compromiso para aprovechar al máximo unos recursos limitados. Si sólo se dispone de seis meses, es imposible construir un sistema ideal desde cero. Si usted tiene que construir un producto que sea compatible con otros ya existentes, no podrá conferirle una arquitectura revolucionaria. Si un producto ha de venderse a un precio determinado, habrá límites para los precios de los componentes a utilizar. Por consiguiente, es tarea del ingeniero diseñar el mejor sistema posible dentro de esas limitaciones.

La ingeniería de hardware se ha constituido en una carrera privativa de los jóvenes. La "crisis de la edad", que suele afectar en la mayoría de las industrias a los trabajadores que tienen entre 40 y 45 años, por lo general se presenta antes entre los ingenieros informáticos. En efecto, muchos de ellos, al acercarse a los treinta años de edad, se ven invadidos por una sensación de fracaso si hasta ese momento no han logrado introducirse en la gerencia o en el departamento de ventas. Con frecuencia el ingeniero de 35 años de edad se suele ver como alguien "venido a menos" que carece de las aptitudes personales y para la comunicación necesarias para la gerencia y cuyos conocimientos tecnológicos probablemente están desfasados.

El acceso a la carrera de ingeniería de hardware se produce a dos niveles. Muchas grandes empresas (en especial aquellas pertenecientes al campo de la defensa) suelen contratar a jóvenes de entre 16 y 18



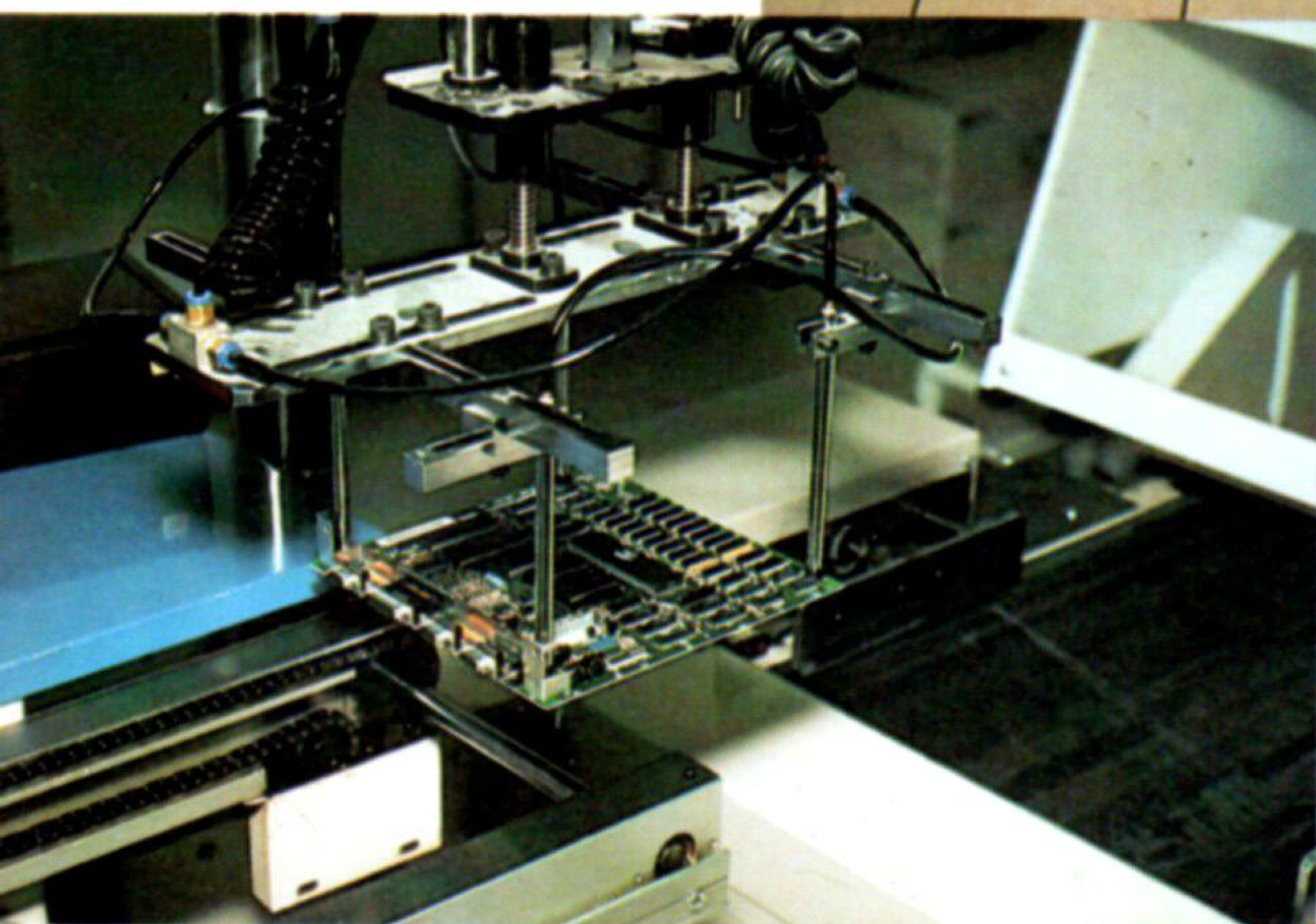
© Science Photo Library

Diseño de placas
La mayoría de los miniordenadores y ordenadores centrales se diseñan de forma modular, lo que facilita la tarea del técnico. Se dedican diferentes placas a las distintas funciones, y se suele disponer de equipos de comprobación para controlar su rendimiento

años de edad con altas calificaciones y proporcionarles formación para que se desempeñen como aprendices. Tal como sucede en la mayoría de las profesiones moderadamente bien remuneradas, la ingeniería de hardware se está volcando cada vez más hacia los graduados. Si usted desea introducirse a nivel de graduado, necesitaría un título en física o bien en ingeniería electrónica; la ingeniería de hardware no es un campo que se preste a las matemáticas, la informática, ni otras titulaciones de cálculo.



La ayuda de la alta tecnología
A los técnicos se les está permitiendo disponer de una creciente cantidad de equipos inteligentes para facilitar el diagnóstico y la corrección de fallos. Aquí vemos a un técnico explorando un gran ordenador central Cray con un detector infrarrojo, en busca de "puntos calientes" que podrían indicar un defecto de funcionamiento. Además, ahora muchos sistemas se diseñan con módulos incorporados de comprobación de fallos fácilmente sustituibles, lo que simplifica el papel del técnico y, en algunos casos, hace casi prescindible su aportación



Diseño automatizado
El papel del ingeniero de diseño suele incluir, en las primeras etapas del desarrollo del producto, la consideración cabal de los medios de construcción. Un producto nuevo destinado a una venta masiva se ha de diseñar de modo de obtener el máximo partido de las técnicas de producción automatizada. La placa del Apple Macintosh que vemos en la fotografía la está ensamblando un robot

En general, las empresas dedicadas a desarrollar proyectos de carácter militar no suelen pagar tan bien como el sector comercial. En Gran Bretaña, es probable que los graduados o aprendices que hayan concluido su preparación sean contratados con sueldos que oscilen entre £8 000 y £10 000. Contrariamente a otras especializaciones informáticas, en ingeniería de hardware es necesario tener una experiencia superior a un año para obtener un aumento salarial de importancia; durante los primeros dos o tres años, los incrementos tienden a ser del orden de apenas un 10 %. El ingeniero de hardware experimentado suele percibir unos emolumentos de entre £10 000 y £13 000 en el campo de la defensa y de entre £13 000 y £16 000 en el sector comercial.

A la edad de entre 28 y 30 años, el ingeniero tenderá a dejar el campo para introducirse en la dirección o las ventas, valiéndose de sus cualificaciones técnicas e industriales. Por supuesto, los salarios pueden ser especialmente elevados en el campo de ventas, percibiendo los vendedores hasta £40 000. La dirección de proyectos es el primer peldaño en la escalera de la gerencia. Tras él, sin embargo, el ingeniero se convertirá en gerente y sus aptitudes técnicas ya no tendrán una importancia tan preponderante.

La ingeniería de software marcha paso a paso con la ingeniería de hardware. Esencialmente, la tarea del ingeniero de software consiste en proporcionar el entorno de software (el sistema operativo) que acompañará al hardware. Con frecuencia, el papel del ingeniero de software es el de adaptar un sistema operativo ya existente aportado por una empresa independiente en lugar de producir un sistema completamente nuevo.

Debido a la escasez de personal que cuente con la suficiente preparación, los ingenieros de software tienden a percibir sueldos más altos que los ingenieros de hardware. En Gran Bretaña, éstos parten de alrededor de £10 000, subiendo hasta £20 000. El acceso se concede casi exclusivamente a los graduados, y, a diferencia de la ingeniería de hardware, se contrata a titulados en carreras de cálculo y se les proporciona adiestramiento.

La crisis que suele afectar a los ingenieros de hardware al acercarse a los treinta años de edad también alcanza a los ingenieros de software, si bien de manera menos acusada. Nuevamente, el ingeniero de software tiende a pasar a las ventas, la gerencia o al asesoramiento.

En general, los ingenieros de hardware pueden pasarse al campo del software, mientras que los ingenieros de software no pueden pasarse al hardware. Los dos papeles laborales tienden a estar separados, sin embargo, colaborando en un proyecto trabajadores de cada campo.

Apoyo técnico

El campo del apoyo técnico proporciona numerosas oportunidades para quienes deseen introducirse en la industria del ordenador y posean alguna experiencia en cuanto a máquinas personales. Al igual que en la ingeniería, existe una clara distinción entre mantenimiento y apoyo de hardware y software. De las dos áreas, el hardware tiende a ser "el pariente pobre", con niveles salariales inferiores a los del área de software.

El apoyo de hardware cubre la instalación de productos en el local del cliente y la comprobación y el diagnóstico de fallos de hardware. El personal de apoyo de hardware tenderá a trabajar para empresas de suministros o bien para empresas de mantenimiento independientes, como las que se contratan para mantener y "detectar problemas" en los ordenadores de otras empresas.

Casi todos los comerciantes de micros de gestión suelen contratar personal de apoyo técnico de hardware y a menudo también a aficionados jóvenes y perspicaces. Los salarios iniciales son bastante bajos, pero con el paso del tiempo pueden alcanzar un nivel bastante elevado.

En el contexto de un negocio pequeño, la persona de apoyo tenderá a adquirir muchísima expe-

Hacia la cúspide

Tanto los ingenieros de hardware como los de software se enfrentan, mediada su carrera, ante una decisión crítica, puesto que las escalas salariales y las oportunidades para progresar son limitadas a menos que el candidato pueda introducirse en el campo de las ventas, la gerencia o el asesoramiento. El diagrama ilustra los distintos caminos existentes, junto con una orientación de los ingresos en el ámbito informativo británico (una libra esterlina equivale a unas 215 pesetas)

Asesor de software £20-25 000

Ventas £20-35 000 +

Gerencia

Gerencia £15 500-£25 000

Jefe de proyectos £12 500-15 000

Empresa de mantenimiento de hardware £12 500

Apoyo de software £7 500

Ingeniería de software £10 000-15 000

Ingeniería de hardware £6 000-£12 000

Formación £5-7 500

Mantenimiento de hardware (empresa de usuario final) £5 000

Graduado

Altas calificaciones

Caroline Clayton

riencia en materia de software, en la demostración de productos y en el asesoramiento al personal de ventas. Una persona joven de apoyo de hardware en un comercio de IBM PC o en una casa de sistemas de miniordenador puede esperar encontrarse frente a varios caminos para hacer carrera: el negocio de apoyo de software algo mejor remunerado, trabajar para una de las casas de mantenimiento independientes o pasar al campo de las ventas.

No obstante, antes de introducirse en el campo del apoyo de hardware, el aspirante ha de tener presente algunos puntos. Si bien en algunas ocasiones no parece así, en realidad los ordenadores se están volviendo cada vez más fiables y se los está diseñando para ser más fáciles de apoyar. En algunos ordenadores, el ingeniero simplemente ha de sustituir un módulo por otro, y los programas de autodiagnóstico significan que se requieren menos conocimientos para averiguar lo que marcha mal en un ordenador. Teniendo presente esto, en un futuro cercano el personal de apoyo de hardware bien podría encontrarse con que su aportación tecnológica sea cada vez más prescindible.

El apoyo de software supone averiguar por qué un software determinado no funciona bien en una máquina determinada y remediarlo. El personal de apoyo de software también habrá de transferir programas de una máquina a otra y podría verse involucrado en áreas especializadas tales como las comunicaciones.

Al igual que en el apoyo de hardware, en este campo se pueden introducir personas con experiencia relativamente escasa. Esencialmente, hay tres niveles de acceso: como aficionado de entre 16 y 17 años de edad que desee aprender y esté dispuesto a percibir un bajo salario durante un par de años; como usuario final con algo de experiencia de programación y utilizando ordenadores de gestión, o como graduado con una titulación en una carrera de cálculo.

Biblioteca pública

Llegados a este punto, centraremos nuestra atención en la biblioteca de instrucciones estándares de E/S

Al igual que muchos otros lenguajes modernos, el c no especifica detalles de entrada/salida (E/S) dentro del lenguaje propiamente dicho. Una especificación exhaustiva de E/S para un lenguaje que se ha de utilizar en una amplia variedad de aplicaciones y en varias máquinas diferentes sería muy restrictiva, dificultando muchas operaciones directas, como es el caso del COBOL y el FORTRAN.

Por ejemplo, en COBOL es imposible aceptar un único carácter directamente desde el teclado sin emplear extensiones no estándares al lenguaje. No obstante, si la E/S no está especificada en absoluto, entonces resulta difícil lograr la portabilidad de programas entre máquinas y sistemas operativos. La entrada por teclado en PASCAL constituye un buen ejemplo de una operación que se puede comportar de modo bastante diferente en distintas versiones del lenguaje.

En c se proporciona un grado de estandarización mediante la biblioteca estándar, stdio.h, que define numerosas funciones de E/S de alto y bajo nivel. Los detalles y las facilidades exactas de la misma pueden variar de una implementación a otra, pero es mejor ceñirse a las funciones de alto nivel. Por otra parte, el programador de una versión específica del lenguaje está en libertad de proporcionar funciones que saquen partido de determinados aspectos de una máquina.

Ya hemos analizado la función "caballo de tiro" para la salida en pantalla (printf) y su correspondiente función para entrada por teclado (scanf). Hay muchas otras funciones que permiten un control más preciso del teclado y la pantalla, así como para E/S a archivos en disco y otros dispositivos. El c ha simplificado la idea de la E/S a archivos y dispositivos tratándolos exactamente de la misma manera. Para el c, un archivo es un flujo de bytes; las funciones de entrada simplemente toman un byte o grupo de bytes de una corriente de entrada, y las funciones de salida envían una corriente. Si se conecta una corriente al disco, es posible desplazarse a lo largo de la corriente hasta una posición especificada, proporcionando de este modo un acceso aleatorio. Cuando los archivos se tratan de esta manera, desaparece la diferencia lógica entre un archivo y un dispositivo físico.

Normalmente los archivos se deben abrir antes de utilizarlos, para especificar el dispositivo o área del disco hacia donde se dirige el flujo de bytes. No obstante, en todo programa en c hay tres archivos que siempre están abiertos. Éstos son: stdin, que normalmente está conectado al teclado; stdout y stderr, que suelen estar conectados a la pantalla. Muchos sistemas operativos, como el Unix, permiten redirigir la entrada y salida, de modo que estas asignaciones se pueden modificar. Las funciones printf y scanf operan sobre stdout y stdin, pero no constituyen más que casos especiales de las funciones más generales fprintf y fscanf.

De modo que, por ejemplo, las sentencias:

```
printf (.....);
fprintf(stdout,.....);
```

son idénticas en operación. Existen otras dos funciones similares, sprintf y sscanf, que realizan E/S hacia y desde series en la memoria principal.

Desde el punto de vista del c, stdout en realidad es un puntero a un dato del tipo FILE, que normalmente se define mediante una macro (# define) en el archivo stdio.h. Para abrir un archivo en disco o dispositivo, se utiliza la función fopen(nombrearchivo,modalidadarchivo). Ésta toma una serie para el nombrearchivo que puede aludir a un dispositivo o archivo según las convenciones del OS. La modalidadarchivo puede ser "r" para leer el archivo, "w" para escribir en él, o "a" para añadirle datos. Si se abre el archivo con "a" o "w" y el mismo no existe, será creado; un archivo que sí exista y se abra con "w" se sobreescribirá. El puntero de archivo (un entero largo que indica la posición actual a lo largo del flujo de bytes) se posicionará al comienzo del archivo con "r" o "w". El valor devuelto por la fun-

Prueba de fuego

Existen muchas funciones y macros útiles incluidas en diversas bibliotecas que nos es imposible analizar. Pero en el archivo ctype.h hay numerosas funciones útiles para manipulación de series. Las mismas se pueden incluir con la línea·

```
# include(ctype.h)
```

El primer grupo proporciona un medio de comprobar determinados atributos en un carácter, devolviendo un valor no cero (verdadero) si el carácter comprobado posee el atributo, y cero en caso contrario. Normalmente están implementadas con gran eficacia.

Nombre	Verdadero si
isalpha(c)	c es una letra
isupper(c)	c está en mayúscula
islower(c)	c está en minúscula
isdigit(c)	c es un dígito
isxdigit(c)	c es un dígito hexa
isspace(c)	c es un carácter espacio
isalnum(c)	c es una letra o un dígito
ispunct(c)	c es un carácter de punt.
isprint(c)	c es un carácter imprimible
iscntrl(c)	c es un carácter de control
isascii(c)	c es un código ASCII

El segundo grupo de funciones proporciona tres convenientes conversiones:

Nombre	Efecto
toupper(c)	convierte c a mayúscula
tolower(c)	convierte c a minúscula
toascii(c)	convierte c a código ASCII



Estudios de biblioteca

Las siguientes son otras funciones de E/S de la biblioteca estándar del C:

getc(puntero_a_archivo) toma el siguiente byte del archivo o dispositivo que se ha abierto con "r" El valor se devuelve como un int. Se devuelve un determinado valor EOF si se encuentra el final del archivo. Puede estar implementada como una macro

getchar() Equivale a getc(stdin)

fgetc(puntero_a_archivo) Vuelve a colocar el carácter c en el archivo desde el cual presumiblemente se había leído. Devuelve el valor int de c. Se debe haber leído al menos un carácter del archivo antes de que se pueda volver a despositar uno, y no es fiable si se coloca más de un carácter por vez

putc(c,puntero_a_archivo) Produce el carácter c en el archivo de salida, devolviendo el valor int de c. Puede ser una macro

putchar(c) Equivale a putc(c,stdout)

fputc(c,puntero_a_archivo) Equivale a putc, pero siempre está implementado como función

gets(s) Donde s es una serie (puntero a char), lee caracteres de stdin hasta encontrar una nueva línea. Esta última no se coloca en la serie, que se terminará correctamente con un "\0". Se devuelve el valor de s

fgets(s,n,puntero_a_archivo) Lee caracteres del archivo en la serie s hasta que se hayan leído n-1 caracteres o hasta hallar una nueva línea. Ésta se colocará en s, que se terminará con "\0". Se devuelve el valor de s

puts(s) Produce la serie s en stdout añadiendo una nueva línea

fputs(s,puntero_a_archivo) Produce la serie s en el archivo sin ninguna nueva línea

fseek(puntero_a_archivo, desplazamiento, lugar) Desplaza el puntero de archivo a lo largo del flujo de bytes hasta el desplazamiento desde el lugar especificado; el lugar puede ser 0 para el comienzo del archivo, 1 para la posición actual o 2 para el final del archivo.
El desplazamiento debe ser de tipo long int

rewind(puntero_a_archivo) Equivale a:
fseek(puntero_a_archivo),0L,0)

ftell(puntero_a_archivo) Devuelve el desplazamiento actual (un long int) desde el comienzo del archivo

unlink(nombrearchivo) Suprime del directorio el archivo mencionado. Devuelve -1 si el archivo no existe, y 0 en caso contrario

exit(estado) Termina un programa devolviendo el valor int de estado al OS o proceso de llamada. Se utiliza 0 para una terminación normal

Caroline Clayton

ción fopen es un puntero al tipo FILE que se puede utilizar en otras funciones; será NULL si por algún motivo no se puede acceder al archivo.

Existe una función fclose(puntero_al_archivo) que cierra un archivo abierto. Todos los archivos se cerrarán automáticamente al terminar un programa, pero esta función podría ser necesaria en razón de un límite del OS sobre la cantidad de archivos que se puedan tener abiertos de forma simultánea.

Tasación de caracteres

```
/* este programa contará el número de palabras, y
el número de caracteres en un archivo, cuyo
nombre se da en la línea de comando */
# include(stdio.h)
# include(ctype.h)
main(argc,argv)
int argc;
char * argv;
  {
  int cont_car=0,cont_pal=0,c,
  inword    =0;
  FILE*en_archivo, * fopen();
/* observe el nombre del archivo y la función
fopen, declarados como punteros al tipo FILE */
/*comprobación de la cantidad correcta de
argumentos*/
  If(argc !=2)
  {
     fprintf(stderr," \nusage es %s
     nombrearchivo\n",
     *argv);
/*recuerde que la primera entrada en una matriz
argv es el verdadero nombre del programa*/
     exit(1);
  }
/*abrir el archivo y comprobar si existe, ++ argv
apunta al nombre del archivo*/
  if((en_archivo=fopen(*++argv,"r"))
  ==NULL)
{
     fprintf(stderr,"\nno puede abrir %s\n",
     *argv);
/*recuerde que ahora argv está apuntando al
nombre del archivo*/
  exist(1);
  }
  while((c=getc(en_archivo)   !=EOF)
  {
     ++cont_car;
     if(enpalabra)
     {
        if(isalnum(c))
        ;/*una sentencia vacía*/
        else
        {
           enpalabra=0;
           ++cont_pal;
        }
     }
     else
        if(isalnum(c))
           enpalabra=1;
  }
  if(enpalabra)
     ++cont_pal;
  printf("\nnúmero de caracteres=%d",
  cont_car);
  printf("\nnúmero de palabras=%d\n",
  cont_pal);
  fclose(en_archivo);
}
```

Opciones abiertas

Analizaremos las herramientas del Unix relacionadas con la gestión de archivos y la preparación de textos

La forma general de un comando Unix es:

nombre_comando opciones argumentos

Cada porción del comando debe estar separada del siguiente por al menos un espacio. Los argumentos de un comando por lo general son nombres de archivo o directorios; si se omite el argumento, se utilizan por defecto los archivos standardinput y standardoutput (el teclado y la pantalla) para la entrada y salida.

Las opciones asumen la forma de letras individuales, precedidas por un guión, y se utilizan en unión con el comando para llevar a cabo una serie de tareas. Allí donde las opciones no requieren otra información, como el nombre de un archivo, tras el guión puede ir más de una opción. Por ejemplo, el comando de listado del directorio ls posee varias opciones, incluyendo l, que produce un listado completo, y a, que lista entradas del archivo. Los argumentos para ls pueden ser ya sea una especificación de archivo o bien un directorio, y de no incluirse ningún argumento se tomará por defecto el directorio actual.

Para listar el contenido del directorio /usr, con ambas opciones seleccionadas, el comando podría ser uno de los siguientes:

ls –l–a/usr
ls –la/usr

Los comandos impartidos incorrectamente generan un mensaje de error, que indica al usuario que no se reconoce el comando o bien, de ser posible, le detalla la utilización correcta. Observe que se pueden escribir en una misma línea dos o más comandos, separándolos mediante un punto y coma.

El comando wc cuenta el número de caracteres, palabras y líneas en un archivo de texto. Las opciones son:

- **l** para contar sólo líneas
- **w** para contar sólo palabras
- **c** para contar sólo caracteres
- **p** para contar páginas (una página tiene 66 líneas)

Los argumentos pueden ser uno o más nombres de archivo. Si se proporciona más de un nombre de archivo, se cuenta cada archivo y se da un total para todos los archivos especificados. Si no se da como argumento ningún archivo, ws da por sentado que la entrada proviene del teclado.

En standardoutput (normalmente la pantalla), head visualiza las primeras líneas de un archivo. La única opción es una que determina la cantidad de líneas a visualizar. Por ejemplo, –15 visualiza las primeras 15 líneas. Los argumentos deben incluir un nombre de archivo o más.

El comando tail da las últimas líneas de un archivo. Las opciones son:

- **+n** una excepción a la regla de que las opciones siempre comienzan con –; se visualiza el resto del archivo a partir de un punto de 10 líneas desde el comienzo
- **–n** visualiza las n líneas finales del archivo; el valor por defecto es 10
- **l** cuenta en líneas (por defecto)
- **b** cambia la unidad a bloques de almacenamiento de disco
- **c** cambia la unidad a caracteres
- **r** visualiza el archivo por orden inverso

La instrucción sort clasifica un archivo por orden de clave, o clasifica y mezcla varios. Las opciones son:

- **b** para ignorar los espacios delanteros
- **d** orden de diccionario, utilizando sólo letras, dígitos y espacios en blanco
- **f** insensible a mayúsculas o minúsculas
- **n** clasificar números por valor aritmético en vez de por dígitos
- **o** dirige la salida a un archivo en lugar de a standardoutput
- **r** clasificar por orden inverso

Los argumentos son uno o más nombres de archivo; si sólo se da un nombre de archivo, se clasifica el contenido de ese archivo. La especificación de múltiples nombres de archivo hace que los archivos se clasifiquen y mezclen entre sí.

El comando cmp compara el contenido de dos archivos para determinar cualquier diferencia entre ambos. Cuando se opera en su modalidad por defecto, el comando devuelve el byte y número de línea en donde se detectó la primera diferencia. La única opción, l, informa sobre todas las diferencias entre los dos archivos. Si se omite un nombre de archivo, se da por sentado standardinput.

La instrucción comm considera dos archivos que se han de clasificar por orden de código ASCII y visualiza tres columnas: aquellas líneas existentes en el primer archivo; aquellas líneas existentes sólo en el segundo archivo; y aquellas líneas comunes a ambos. Las opciones son, simplemente, 1, 2 y 3 para omitir una de las tres columnas. Se deben especificar como argumentos los dos nombres de archivo.

La instrucción diff también encuentra las diferencias entre dos archivos. Indica los cambios que se deben efectuar en el primer archivo para hacerlo idéntico al segundo, utilizando a para añadir, c para cambiar, d para suprimir, < para una línea del primer archivo, y > para una línea del segundo archivo. Las opciones son:

- **b** para ignorar espacios en blanco finales e igualar las series de espacios en blanco independientemente de su longitud
- **e** para producir salida como instrucciones para el editor
- **r** sólo se emplea con directorios, y permite que diff se aplique a sí misma de forma recursiva para cualquier subdirectorio

Los argumentos pueden ser tanto un par de nombres de archivo como de nombres de directorio. Si los argumentos especifican dos directorios, diff lista todos los archivos exclusivos de cada directorio, y

luego lista en la tercera columna aquellos archivos comunes a ambos.

El comando uniq compara líneas adyacentes de un archivo de texto y suprime las entradas repetidas. Las opciones son:

- **u** para visualizar sólo aquellas líneas que no se repiten
- **d** para visualizar sólo las líneas que se repiten
- **c** para acompañar cada línea de salida con el número de veces que aparece dentro del archivo

Los argumentos pueden ser uno o dos nombres de archivo. Si se especifican dos archivos, la salida pasa al segundo archivo mencionado, en lugar de a la pantalla (standardoutput).

El comando lpr envía uno o más archivos a la impresora del sistema. Dado que la salida desde trabajos multitarea no se puede realizar directamente a la impresora, se coloca en una cola. El Unix explora la cola continuamente e imprime la salida de archivo en la cabeza de la cola, que se conoce como *spooling*. Los argumentos son uno o más nombres de archivo. No hay ningún juego de opciones estándares, puesto que las facilidades de impresión varían entre distintas instalaciones.

El comando lpq visualiza detalles del estado actual de la cola de impresión, que permiten comprobar si se ha imprimido un archivo determinado, o durante cuánto tiempo permanecerá en la cola a la espera de su turno de impresión. A cada tarea de impresión de la cola se le asigna un número, y no hay argumentos ni opciones estándares.

El comando lprm permite sacar un archivo de la cola de impresión antes de que se lo imprima. El argumento puede ser ya sea un nombre de archivo, el número de tarea de impresión obtenido mediante el uso de lpq, o un nombre login, en cuyo caso se suprimirán todos los archivos que pertenezcan a ese propietario.

El comando pr visualiza el contenido del archivo en standardoutput, formateado para impresión. El texto se organiza en páginas, con cinco líneas de margen inferior y un encabezamiento, compuesto por la fecha, el nombre del archivo y el número de página seguido por dos líneas en blanco. Por lo general, la salida se entuba en lpr. Las opciones son:

- **n** para organizar el texto en un número dado de columnas
- **m** para visualizar dos o más archivos lado a lado
- **t** para suprimir el encabezamiento y el margen inferior

Los argumentos son uno o más nombres de archivo. A continuación vemos un ejemplo de la potencia y eficacia del Unix ante la gestión de tres archivos de texto.

Gestión de archivos

```
%cat file1
The cat sat on the mat.
Mary had a little lamb.
The quick brown fox jumps over the lazy dog.
The owl and the pussy cat went to sea.

%cat file2
The cat sat on the dog.
Mary had a little lamb.
The quick brown fox jumps over the lazy dog.
The owl and the pussy cat went to sea.

%cat file3
The cat sat on the mat.
Mary had a little lamb.
The quick brown fox jumps over the lazy dog.
This file has an extra line.
The owl and the pussy cat went to sea.

%wc file1        {contar el número de líneas, palabras y caracteres}
      4    29    132 file1

%wc -w file1     {contar sólo palabras}
    29 file1

%head -2 file1   {visualiza sólo las dos primeras líneas}
The cat sat on the mat.
Mary had a little lamb.

%tail +2 file1   {visualiza desde la segunda línea}
Mary had a little lamb.
The quick brown fox jumps over the lazy dog.
The owl and the pussy cat went to sea.

%tail –2 file1   {visualizar las dos últimas líneas}
The quick brown fox jumps over the lazy dog.
The owl and the pussy cat went to sea.

%sort file1      {clasificar archivo por orden}
Mary had a little lamb.
The cat sat on the mat.
The owl and the pussy cat went to sea.
The quick brown fox jumps over the lazy dog.

%pr file3        {formatear un archivo para impresión}
The cat sat on the mat.
Mary had a little lamb.
The quick brown fox jumps over the lazy dog.
This file has an extra line.
The owl and the pussy cat went to sea.

%sort file1 > file 4       {crear versiones clasificadas usando redirección}

%sort file2 > file5

%cmp file1 file2           {comparar un par de archivos}
file1 file2 differ char 20, line 1

%comm file4 file5
                                    Mary had a little lamb.
                 The cat sat on the dog.
The cat sat on the mat.
                                    The owl and the pussy cat went to sea.
                                    The quick brown fox jumps over the lazy dog.
{lines en file4/lines en file5/lines en ambos archivos}

%diff file1 file2          {visualizar diferencias entre archivos}
1c1                        {"c" implica "efectuar un cambio"}
< The cat sat on the mat.
---
> The cat sat on the dog.

%sort file2 file3 > file7  {mezclar dos archivos}
%cat file7
Mary had a little lamb.
Mary had a little lamb.
The cat sat on the dog.
The cat sat on the mat.
The owl and the pussy cat went to sea.
The owl and the pussy cat went to sea.
The quick brown fox jumps over the lazy dog.
The quick brown fox jumps over the lazy dog.
This file has an extra line.

%uniq file7                {visualizar archivo ignorando repeticiones}
Mary had a little lamb.
The cat sat on the dog.
The cat sat on the mat.
The owl and the pussy cat went to sea.
The quick brown fox jumps over the lazy dog.
The file has an extra line.
```

Cacería de patos

Su microordenador le invita a participar en una cacería de patos. Déjese tentar por este juego escrito por Pierre Monsaut para el Commodore 64

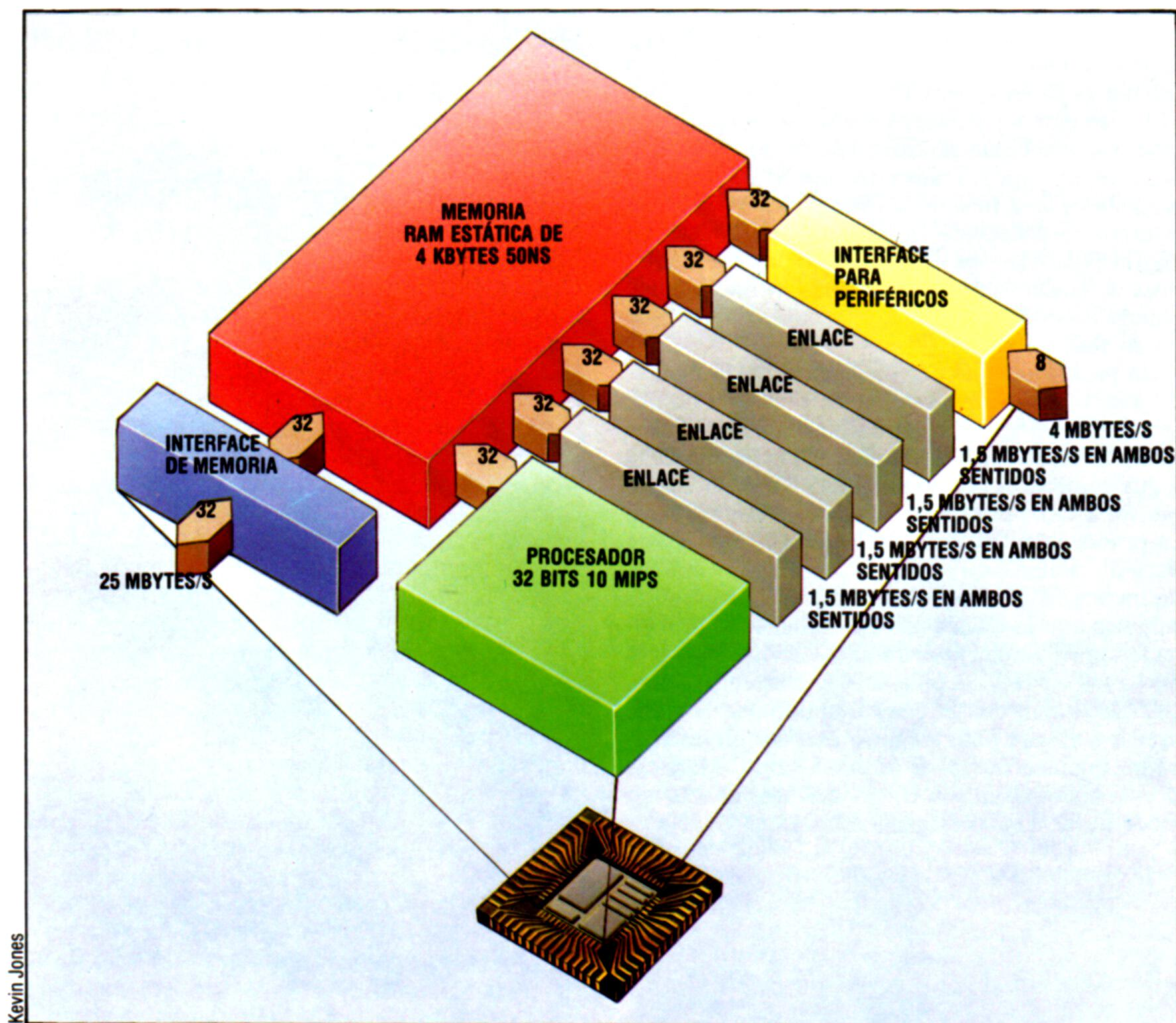


Los patos vuelan de derecha a izquierda por la parte alta de la pantalla. Usted se desplaza utilizando las teclas de control del cursor. Puede efectuar tantos disparos como quiera, pero tan sólo pasarán 20 patos y su objetivo es abatir el mayor número posible. Para disparar pulse la barra espaciadora. Si alcanza un pato, conseguirá un punto y lo verá caer batiendo sus alas. En esta versión el juego resulta bastante sencillo. Si desea aumentar su dificultad, sustituya la línea 4000 por:

```
4000 IF T-C = 1025 THEN 5000
```

```
5 REM ********************
10 REM * LA CAZA DEL PATO *
15 REM ********************
20 GOSUB 1000
100 X=-X
110 Y=X+0.5
120 C=C-1
130 IF C<0 THEN GOSUB 3000
140 IF NC=0 THEN 4500
150 PRINT CL$;TAB(C);CC$(INT(Y))
160 PJ=PJ+D1
170 IF PJ<PN THEN PJ=PN
180 IF PJ>PM THEN PJ=PM
190 POKE P1,CR
200 POKE PJ,CJ
210 POKE PJ+M,JC
220 P1=PJ
230 IF ABS(T-PJ)>2 THEN T=T-80:GOTO 250
240 T=PJ
250 IF T<1064 THEN 4000
260 POKE T,CJ
270 POKE T+M,CJ
280 IF T<>PJ THEN POKE T+80,CR
290 GET X$
300 D1=2*((X$=G$)-(X$=D$))
310 IF X$<>"[1SPC]" OR T<>PJ THEN 100
330 T=PJ-80
340 GOTO 100
1000 DIM CC$(1)
1010 CJ=30
1020 JC=5
1030 CC$(0)=CHR$(173)+CHR$(113)+CHR$(189)
     +"[1SPC]"
1040 CC$(1)=CHR$(176)+CHR$(113)+CHR$(174)
     +"[1SPC]"
1050 PJ=2004
1060 NC=20
1070 T=PJ
1080 X=0.5
1090 C=37
1100 M=54272
1110 CL$=CHR$(19)
1120 PM=2022
1130 PN=1985
1140 CR=32
1150 P1=PJ
1160 D$=CHR$(29)
1170 G$=CHR$(17)
2000 PRINT CHR$(147);
2010 POKE 53280,0
2020 POKE 53281,0
2030 PRINT CHR$(30);
2040 RETURN
3000 PRINT CL$;"[3SPC]";
3020 FOR I=1 TO 200
3030 NEXT I
3040 C=37
3050 NC=NC-1
3060 RETURN
4000 IF ABS((T-1024)-(C+1))<=1 THEN 5000
4010 POKE T+80,CR
4020 T=PJ
4030 IF NC<>0 THEN 260
4500 PRINT CHR$(147)
4510 FOR I=1 TO 10
4520 PRINT
4530 GET X$
4540 NEXT I
4550 PRINT TAB(13)"PUNTOS[1SPC]:";S
4560 FOR I=1 TO 10
4570 PRINT
4580 NEXT I
4590 PRINT TAB(16)"OTRA[1SPC]?"
4600 GET X$
4610 IF X$="" THEN 4600
4620 IF X$<>"N" THEN RUN
4630 END
5000 S=S+1
5010 D=C
5020 C=37
5030 PRINT CL$;
5040 FOR I=1 TO 22
5050 X=-X
5060 Y=X+0.5
5070 PRINT TAB(D);"[3SPC]"
5080 PRINT TAB(D);CC$(INT(Y))
5090 PRINT CHR$(145);
5100 FOR J=1 TO 50
5110 NEXT J
5120 NEXT I
5130 NC=NC-1
5140 FOR I=1 TO 500
5150 NEXT I
5160 PRINT CHR$(147);
5170 IF NC=0 THEN 4500
5180 GOTO 100
```



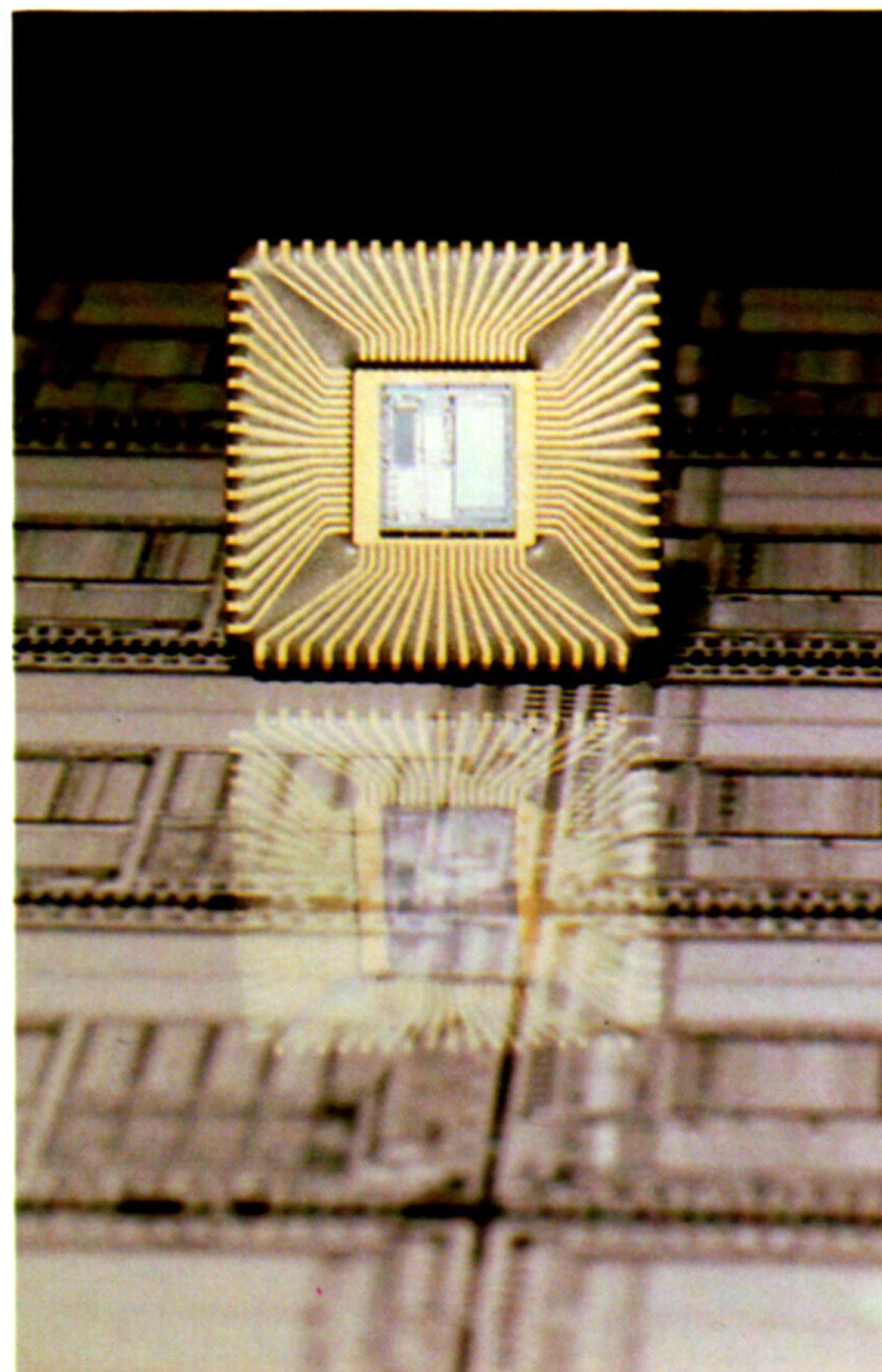


Cálculo trascendental
El Transputer, un nuevo desarrollo de Inmos, es un ordenador en un chip, equipado con un procesador de 32 bits, RAM y cuatro enlaces E/S en serie. El Transputer se considera como un componente a utilizar dentro de la arquitectura de ordenadores en paralelo mucho mayores. Los enlaces en serie permiten conectar entre sí en red varios Transputers, la topología de la cual vendrá determinada por la tarea para la que se haya diseñado el sistema

Ordenador en un chip

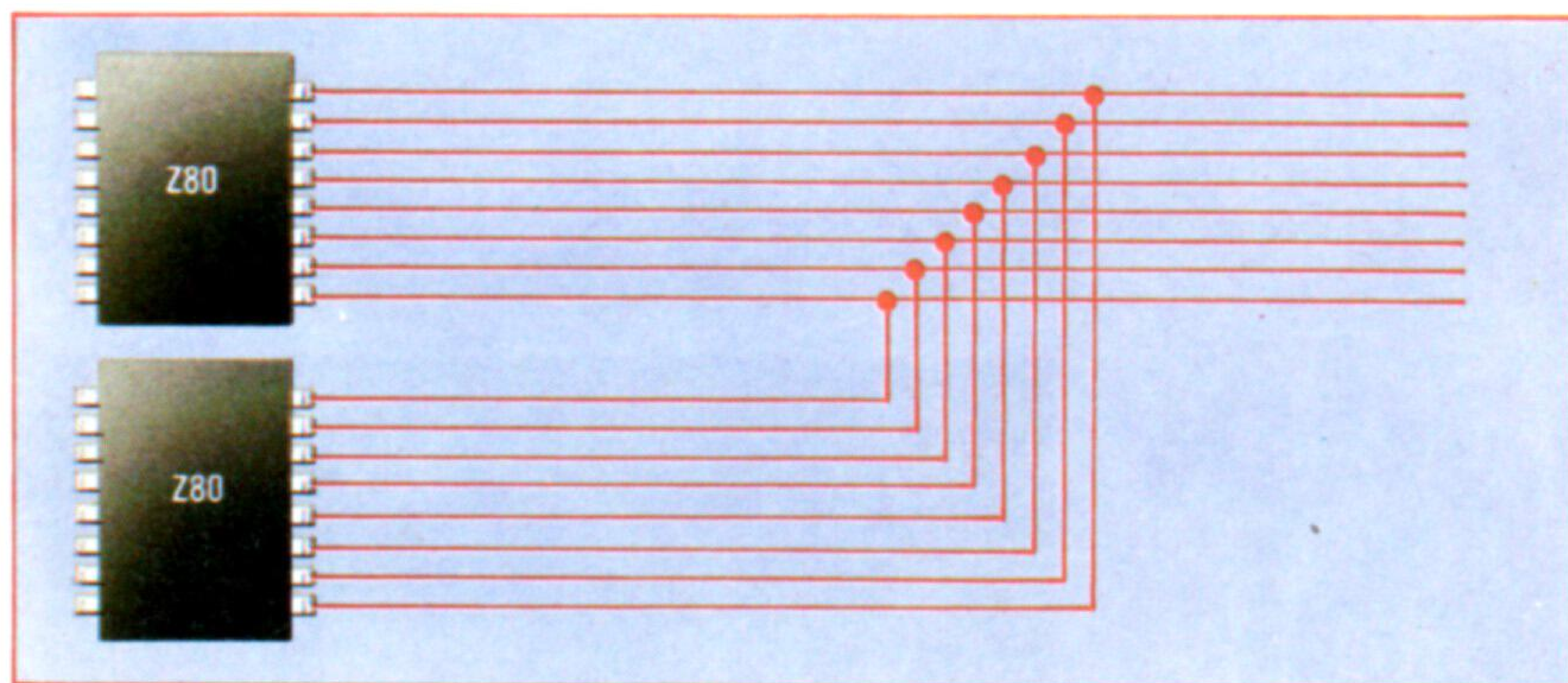
El lanzamiento del Inmos Transputer probablemente igualará o superará el impacto que supuso la irrupción del transistor en los años cincuenta

La invención del transistor en los años cincuenta abrió el camino para la fabricación masiva de ordenadores. Los microprocesadores de hoy en día se construyen a partir de centenares de miles de interruptores de transistor idénticos en el mismo trozo de silicio. El Inmos Transputer es un nuevo y revolucionario microprocesador destinado a utilizarse de forma similar al transistor: como un componente o bloque de construcción para sistemas más grandes (el nombre sugiere un híbrido de *trans*istor y com*puter*). Se pueden construir ordenadores en paralelo mediante la combinación de grandes cantidades de Transputers, que son tratados como "com-

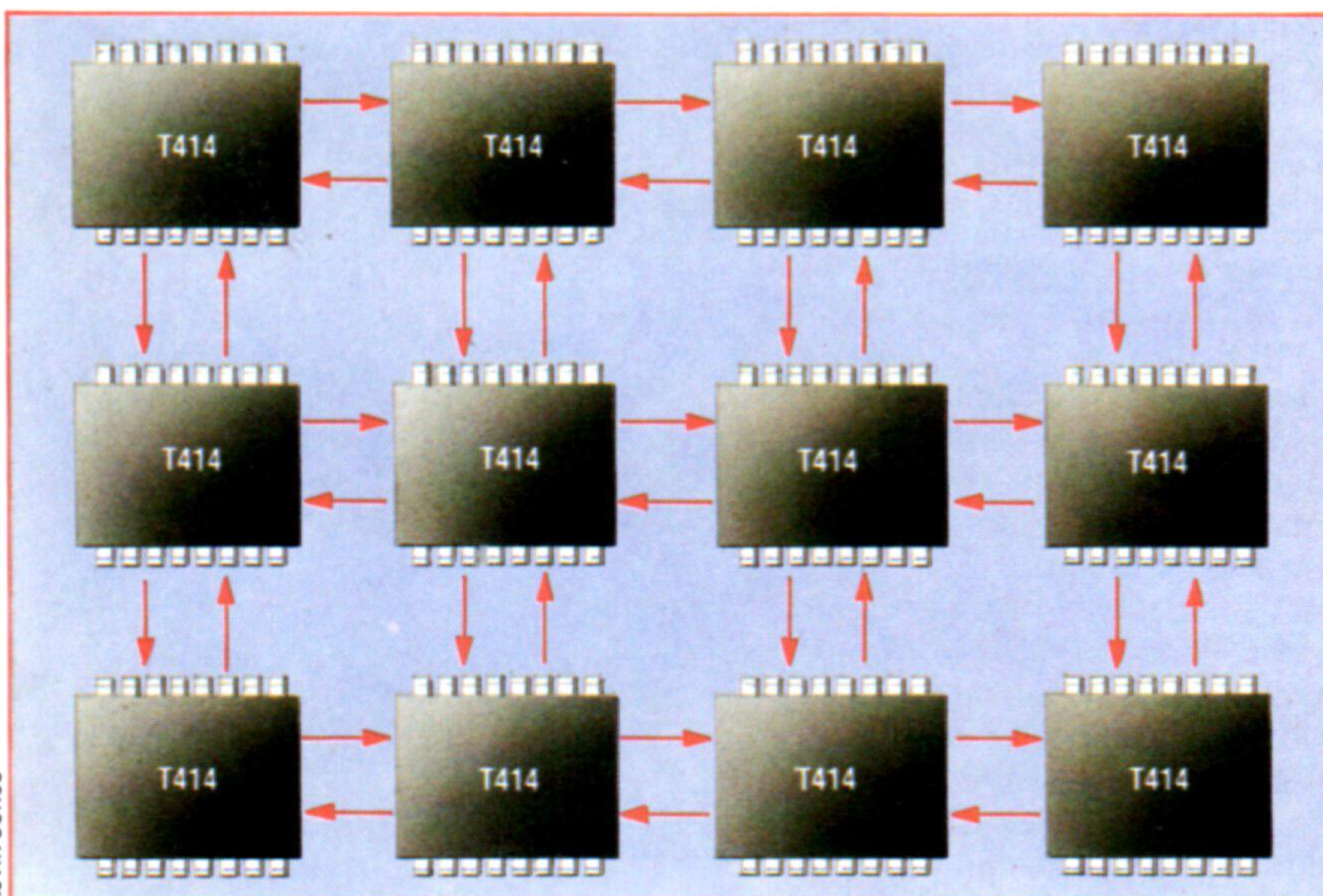


ponentes programables" en vez de como CPU individuales y omnipotentes, ya que el Transputer incluye en un único chip todos los componentes de un sistema de ordenador. Cada Transputer posee una CPU, memoria y enlaces para comunicaciones en serie, todo en el mismo trocito de silicio. Ello significa que un único Transputer puede ejecutar un programa por sí mismo, sin necesidad de recursos externos (a excepción de potencia eléctrica y un reloj). Además, los Transputers pueden hablar entre sí mucho más fácilmente de lo que pueden hacerlo las CPU tradicionales, a través de sus enlaces en serie.

Un microprocesador tradicional, como el Z80, es el "jefe" en cualquier sistema en el cual se utilice. La comunicación con el mundo exterior se realiza a través de un bus en paralelo, siendo la propia CPU la que controla el acceso al mismo. Es difícil lograr que varios Z80 se hablen entre sí por dos razones. En primer lugar, físicamente es difícil diseñar una placa de circuitos en la cual muchos chips compartan un bus en paralelo; se necesita unir ocho cables cada vez que se realiza una conexión. Este problema se agrava, en lugar de remitir, con los chips más modernos de 16 y 32 bits. En segundo lugar, cada Z80 desea controlar el bus, de modo que se debe diseñar software para asegurar que dos procesadores no intenten hablar al mismo tiempo. Mientras se está comunicando a través del bus, el Z80 no puede llevar a cabo ninguna otra tarea.



Como cada Transputer posee cuatro enlaces en serie incorporados, puede hablar a otros cuatro chips.



Kevin Jones

Arquitectura Von Neumann

Todos los ordenadores que utilizamos en la actualidad se basan en los principios enunciados por John von Neumann en 1945. Un ordenador Von Neumann consta de una unidad central de proceso (CPU), conectada a alguna memoria que retiene tanto las instrucciones del programa que especifican lo que se ha de hacer como los datos sobre los cuales se hará. La CPU busca las instrucciones una después de la otra, o secuencialmente, y las ejecuta.

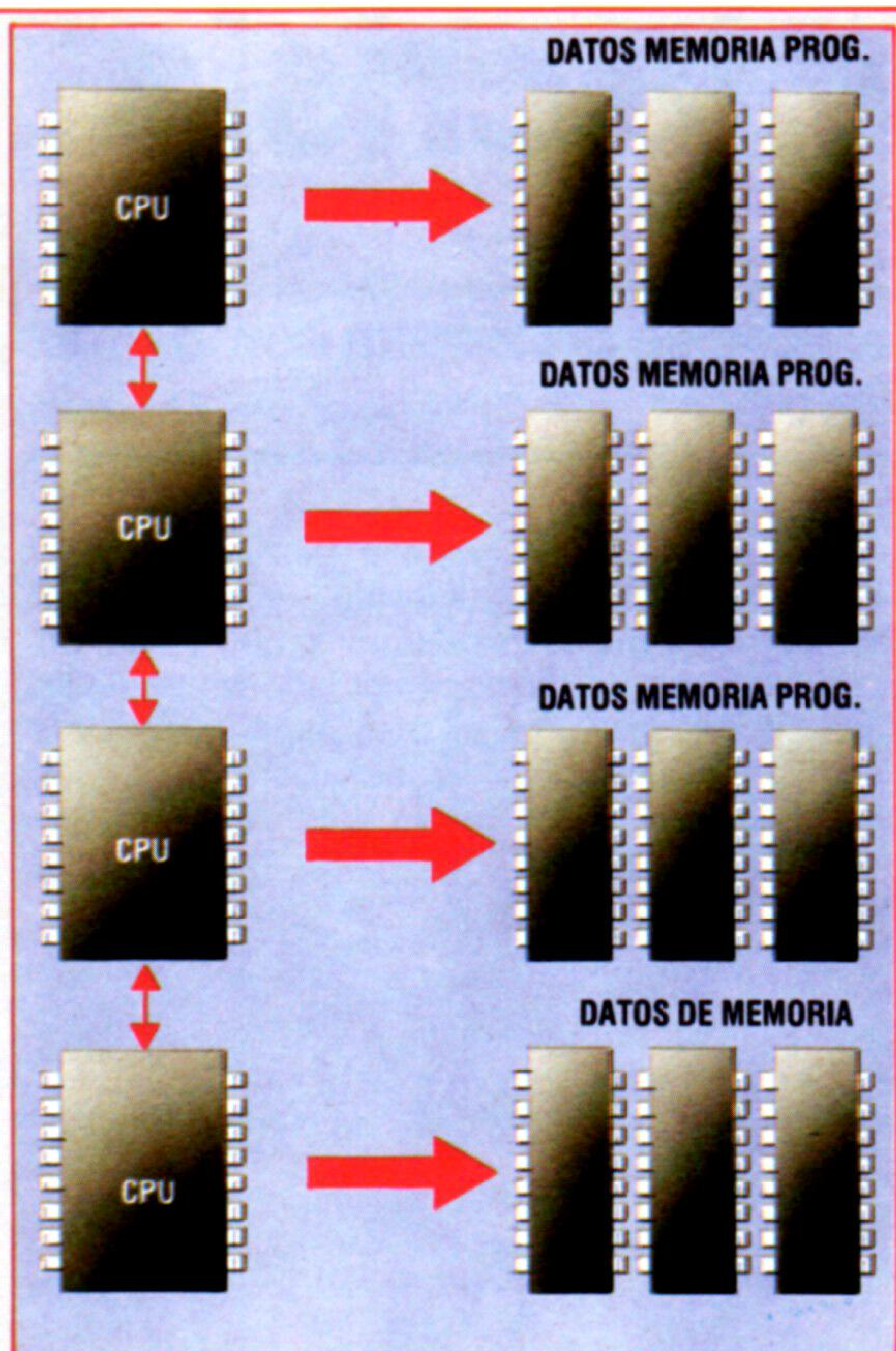
La arquitectura Von Neumann hizo posible que los ordenadores para fines generales se convirtieran en una propuesta práctica. La idea clave, de que los números en la memoria pueden representar tanto un programa como datos, liberó a los diseñadores que habían construido ordenadores exclusivos para tareas determinadas: descifrado de códigos o control de maquinarias. El problema radica en la forma secuencial en que una máquina Von Neumann busca y ejecuta sus instrucciones. Por más rápido que hagamos operar la CPU, no puede trabajar con mayor celeridad de lo que tarde en buscar sus instrucciones y datos en la memoria, y existen límites físicos en cuanto a la rapidez con que puede realizarlo. Así, la velocidad de comunicación entre un único procesador secuencial y su memoria se ha convertido en un cuello de botella. Se puede evitar diseñando ordenadores con más de una CPU y haciendo que todos trabajen al mismo tiempo, lo que se conoce como *proceso en paralelo*. Aun cuando la velocidad de comunicación con la memoria sea limitada, cien CPU que hablen con cien memorias pueden realizar en un tiempo dado cien veces el trabajo de que es capaz un solo procesador. Los ordenadores en paralelo ganan aún más velocidad porque cada CPU necesita hablar a menos memoria, y esto incrementa la velocidad máxima de comunicación, como vemos en el diagrama. El proceso en paralelo presenta dificultades que han impedido que se generalice su adopción. Para ejecutar un único programa en varias CPU, a cada procesador se le debe dar una parte de la tarea a cumplir, y debe realizar su parte sin entrar en colisión ni en competencia con sus vecinos. Si quisiéramos utilizar dos procesadores para calcular el valor de la expresión:

Esto permite construir matrices de dos o tres dimensiones de Transputers con un cableado mínimo.

Lo más importante, sin embargo, es que los enlaces del Transputer están diseñados teniendo en cuenta las comunicaciones, y no se requiere ningún software especial para evitar las colisiones. Un único Transputer es capaz de realizar muchas tareas al mismo tiempo; en particular, puede recibir mensajes desde sus cuatro enlaces simultáneamente sin tener que interrumpir lo que esté haciendo en ese momento.

Estas características de diseño permiten descomponer en partes un único programa y ejecutarlo en varios Transputers. Cada chip puede ejecutar su propia parte del programa en su propia memoria e informar a sus vecinos sobre los resultados enviándoles mensajes a través de los enlaces. Si necesita ejecutar más rápido una parte determinada del programa, sólo necesita añadir más Transputers.





(3+4)*(9 – 7+3)

cada uno de ellos tomaría una de las expresiones entre paréntesis y la calcularía, y luego uno de ellos realizaría la multiplicación final.

Es esencial prever cuál de las CPU llevará a cabo el paso final. Asimismo, la CPU 1 no debe intentar realizar el último paso hasta que la CPU 2 haya terminado con su parte del cálculo.

Es necesario que las CPU se hablen y se pasen resultados parciales y señales. Esta clase de problemas no se pueden tratar con facilidad mediante los lenguajes tradicionales, que dan por sentado que todo sucederá secuencialmente. Los intentos del pasado en cuestión de ordenadores en paralelo con frecuencia han resultado decepcionantes en rendimiento

Sin precedentes

Tecnología: El T414 se fabrica en tecnología CMOS de 1,5 micrones. Ello lo convierte en uno de los más densos y más complejos que se hayan construido nunca

Procesador: Un microprocesador de 32 bits capaz de ejecutar 10 millones de instrucciones por segundo (aproximadamente 10 veces más rápido que el Motorola 68000, p. ej.). Utiliza sólo 70 instrucciones que soportan el lenguaje OCCAM

Memoria en chip: RAM estática con un tiempo de acceso de 50 nanosegundos. Se utiliza para ejecutar pequeños procesos OCCAM; sustituye a los registros de un microprocesador convencional

Memoria fuera del chip: Se pueden direccionar hasta cuatro gigabytes de memoria externa, permitiendo utilizar el Transputer como chip CPU independiente en ordenadores personales. El procesador no ve diferencias entre memoria en chip y ajena al chip, salvo la velocidad de acceso

Enlaces en serie: Cada enlace es capaz de transferir 10 megabits por segundo en ambas direcciones; los cuatro enlaces pueden operar simultáneamente, dando una velocidad de datos total potencial de 80 Mbits por segundo. La comunicación es asincrónica, con una sincronización por hardware, de modo que los Transputers a cada extremo de un enlace no necesitan compartir la misma señal de reloj

Concurrencia: Un único T414 puede ejecutar múltiples procesos al mismo tiempo, controlado por un planificador de procesos incorporado. Todas las partes del chip operan concurrentemente entre sí, de modo que la parte del procesador puede realizar cálculos o acceder a la RAM mientras se envían mensajes en los enlaces

Otros productos: El T414 es el primero de una familia de productos Transputer compatibles en los que diversas funciones para fines especiales reemplazarán parte de la memoria o enlaces. Dos de ellos ya diseñados son el G213 Graphics Processor y el M212 Disk Controller

La empresa: Inmos se creó en 1978 con la ayuda del gobierno británico y ahora forma parte del grupo Thorn EMI. El diseño y desarrollo del Transputer y el OCCAM se realizó en Bristol, y el Transputer se fabrica en Newport (Gales)

La clave que permite la utilización del Transputer de este modo reside en el lenguaje utilizado para programarlo. El Transputer no posee un lenguaje ensamblador como los microprocesadores tradicionales, sino que ejecuta un lenguaje diseñado *ad hoc* denominado OCCAM, que está confeccionado especialmente para escribir programas paralelos. Un programa en OCCAM se divide en partes llamadas *procesos*, que son algo así como subrutinas o procedimientos en BASIC o en PASCAL. La gran diferencia es que los procesos se pueden ejecutar al mismo tiempo, así como en serie.

Además de almacenar valores como variables, como un lenguaje "convencional", el OCCAM puede comunicar valores a través de canales. De modo que diferentes procesos que se están ejecutando al mismo tiempo pueden comunicarse los resultados entre sí a través de un canal. Además, los canales OCCAM se sincronizan automáticamente, de modo que es imposible que se produzca la comunicación hasta que ambas partes no estén preparadas. Asimismo, el OCCAM posee la propiedad vital de no discernir si los procesos que componen un programa se están ejecutando en el mismo Transputer o en Transputers distintos; un canal puede no ser más que un "buzón" de la memoria en el primer caso, o un trozo de cable en el segundo.

El Transputer ofrece un enfoque nuevo y radical al diseño de sistemas de ordenador de gran rendimiento. Hace borrosa la distinción entre diseño de hardware y diseño de software, porque todo lo que se puede describir mediante un proceso en OCCAM se puede implementar en un chip. Alcanzar objetivos de rendimiento se convierte en una cuestión de elegir el tamaño y la topología (o "forma") adecuados de la red de Transputers para abordar el problema. Quizá el futuro ofrezca anillos, *donuts*, cubos y otras configuraciones.

Para superar la crisis

Los lenguajes "funcionales" como el HOPE se presentan como una posible solución a la crisis del software

Muchos científicos de ordenadores están experimentando con lenguajes de programación *funcionales* como una salida para la "crisis de software" (véase la página contigua). Los programas escritos en lenguajes funcionales tienen la propiedad de que se puede comprender cualquier parte del programa sin remitirse al resto de éste; no dependen de la historia u orden de ejecución de sus partes.

Esta "transparencia" se obtiene mediante la supresión del uso de asignación a variables. Algunos lenguajes funcionales prohíben directamente las variables y emplean "funciones", que devuelven valores para uso inmediato. El principio se puede ilustrar incluso en BASIC mediante estos dos programas:

```
10 X=57
20 A=SIN(X)
30 B=LOG(A)
40 C=SQRT(B)
50 PRINT C

10 PRINT SQRT(LOG(SIN(57)))
```

En el segundo programa no se utilizan variables, pero el resultado es el mismo que en el primero. El programa se compone exclusivamente de la aplicación de funciones a los resultados de otras funciones.

El único lenguaje funcional conocido es la forma "pura" original del LISP, pero la mayoría de los dialectos modernos han añadido características no funcionales, tales como la asignación con SETQ y bucles para acelerar la ejecución en ordenadores convencionales.

Un buen ejemplo de un lenguaje moderno puramente funcional es HOPE, creado en la Universidad de Edimburgo. Los programas en HOPE se escriben definiendo funciones, como en LISP. Cada función consta de una serie de ecuaciones, que indican a la función qué valor devolver para cada forma posible de sus argumentos.

En los programas en HOPE se admiten variables, pero sus valores no se pueden cambiar por asignación; la única forma de otorgar un valor a una variable es utilizándola como un *patrón* que concuerde con los argumentos de la función. Por ejemplo, una función para calcular el cuadrado de un número se podría escribir así:

```
  dec cuadrado: num—> num;
---cuadrado(x)<=x*x;
```

donde dec alude a *declarar* y da comienzo a la definición; num—>num dice que la función cuadrado toma un número como su argumento, y devuelve uno con su valor (en HOPE, como en PASCAL, los valores poseen un tipo). La ecuación (iniciada mediante ---) dice que el cuadrado de cualquier número es ese número multiplicado por sí mismo; el símbolo <= significa "se define como" o "podría sustituirse por". El patrón x del lado izquierdo se empareja con cualquier número dado como argumento. El lado derecho de una ecuación debe ser una expresión, y puede utilizar solamente variables tomadas del patrón.

Nosotros utilizamos la función entrando, pongamos por caso, cuadrado(4); y HOPE responde con 16: num, dando el tipo así como el valor del resultado. Por supuesto, esta función se puede emplear para definir otras funciones. Un ejemplo algo más complejo es una función en HOPE para calcular el factorial de un número:

```
    dec fact:num— >num;
--- fact(0)          <=1;
--- fact (succ(n))   <=(succ(n)*fact(n));
```

Las dos ecuaciones definen el valor de la función para todos los casos posibles (el tipo num representa enteros positivos, de modo que no se plantea el caso negativo). En el caso de que su argumento sea 0, entonces devuelve el valor 1. En cualquier otro caso, el factorial de "uno más que n" es "una más que n" veces el factorial de n. Ésta es una definición recursiva, porque fact está definida desde el punto de vista de ella misma; los lenguajes funcionales utilizan la recursión en lugar de la iteración.

El orden de las ecuaciones es irrelevante, y el siguiente funciona igualmente bien:

```
    dec fact:num—> num;
--- fact(succ(n))<=(succ(n)*fact(n));
--- fact(0)<=1;
```

La función succ (*successor*: sucesor), que devuelve un número de uno más que su argumento, está incorporada en el HOPE y se la conoce como una función *constructora*.

En HOPE, todos los tipos de datos se construyen a través de su función constructora. Cuando escribimos una constante como 3, estamos evaluando una función llamada 3 cuyo valor es 3, pero que en realidad constituye una versión abreviada de expresión succ(succ(succ(0))).

Para representar múltiples objetos del mismo tipo, el HOPE utiliza *listas* en lugar de matrices. Las listas se escriben entre corchetes, de modo que [1,2,3,4] es una lista de cuatro números. Una lista se puede emparejar con un patrón, como x::y, donde x se empareja con el primer elemento de la lista e y se empareja con todo el resto. Utilizando la lista [1,2,3,4], x sería 1 e y la lista [2,3,4]. El símbolo ::, pronunciado "cons", es la función constructora para listas.

Las series de texto se representan como listas de caracteres y alternativamente se pueden escribir

entre comillas, de modo que "juan" significa lo mismo que ['j','u','a','n']. Una función para contar la cantidad de letras de una palabra podría ser:

```
    dec contadorletras:list char—>num;
--- contadorletras(nil)<=0;
--- contadorletras(x::y)<=contadorletras(y)+1:
```

donde nil significa una lista vacía. Se utiliza así:

```
contador letras("aardvaark");
9:num
```

En HOPE los tipos son mucho más flexibles que en PASCAL, y es posible escribir funciones que puedan trabajar sobre cualquier tipo. Por ejemplo:

```
    typevar:alpha

    dec contadorlista:list(alpha)—>num;
--- contadorlista(nil) <=0;
--- contadorlista(x::y) <=contadorlista(y)+1;
```

contará los elementos de una lista de cualquier tipo y se podría utilizar en lugar de contadorletras.

Los programadores pueden definir sus propios tipos de datos, de complejidad cualquiera, y éstos se pueden pasar como argumentos o devolver como valores desde funciones. Incluso es posible pasar funciones como argumentos y devolverlas como resultados, lo que permite escribir programas sumamente potentes.

La crisis de software

La "crisis de software" es una forma drástica de describir el hecho de que, mientras que los ordenadores se vuelven cada vez más veloces y baratos, el costo de escribir software para ellos continúa aumentando y, lo que es aun peor, también aumenta el costo del mantenimiento del software existente.

Las raíces de la crisis de software están en la naturaleza de los lenguajes de programación convencionales. Si bien lenguajes modernos estructurados como el PASCAL han representado alguna mejora, todavía resulta difícil comprender lo que hace un programa simplemente leyendo su código fuente. En el caso del lenguaje ensamblador y lenguajes no estructurados, como el BASIC o el FORTRAN, el problema reviste aún mayor gravedad. Incluso puede ser que el propio autor de un programa tenga problemas para leer su trabajo más adelante, mientras que las otras personas que han de mantener tales programas tienen ante sí una tarea colosal. Respecto a los grandes sistemas de software, como los utilizados en la exploración espacial o en las instalaciones de defensa, es dudoso que alguien comprenda el programa en su totalidad. Gran parte del problema de la lectura de los programas surge del hecho de ser "dependientes de la historia". A diferencia de una fórmula matemática, el texto de un programa para ordenador no siempre transmite toda la información requerida para entender lo que hace. En un programa convencional, el valor de una variable depende de la "historia" previa de ejecución del programa. Tomemos a modo de ejemplo este programa en BASIC BBC:

```
 10 FLAG%=0
 15 REM
 20 DEF FNx(A%)
 30    FLAG%=1
 40    =2*A%
 45 REM
 50 DEF FNy(A%)
 60    LOCAL B%
 70    IF FLAG% THEN B%=3 ELSE B%=4
 80    =B%*A%
 85 REM
 90 PRINT FNy(2)+FNx(1)
100 PRINT FNy(2)+FNx(1)
```

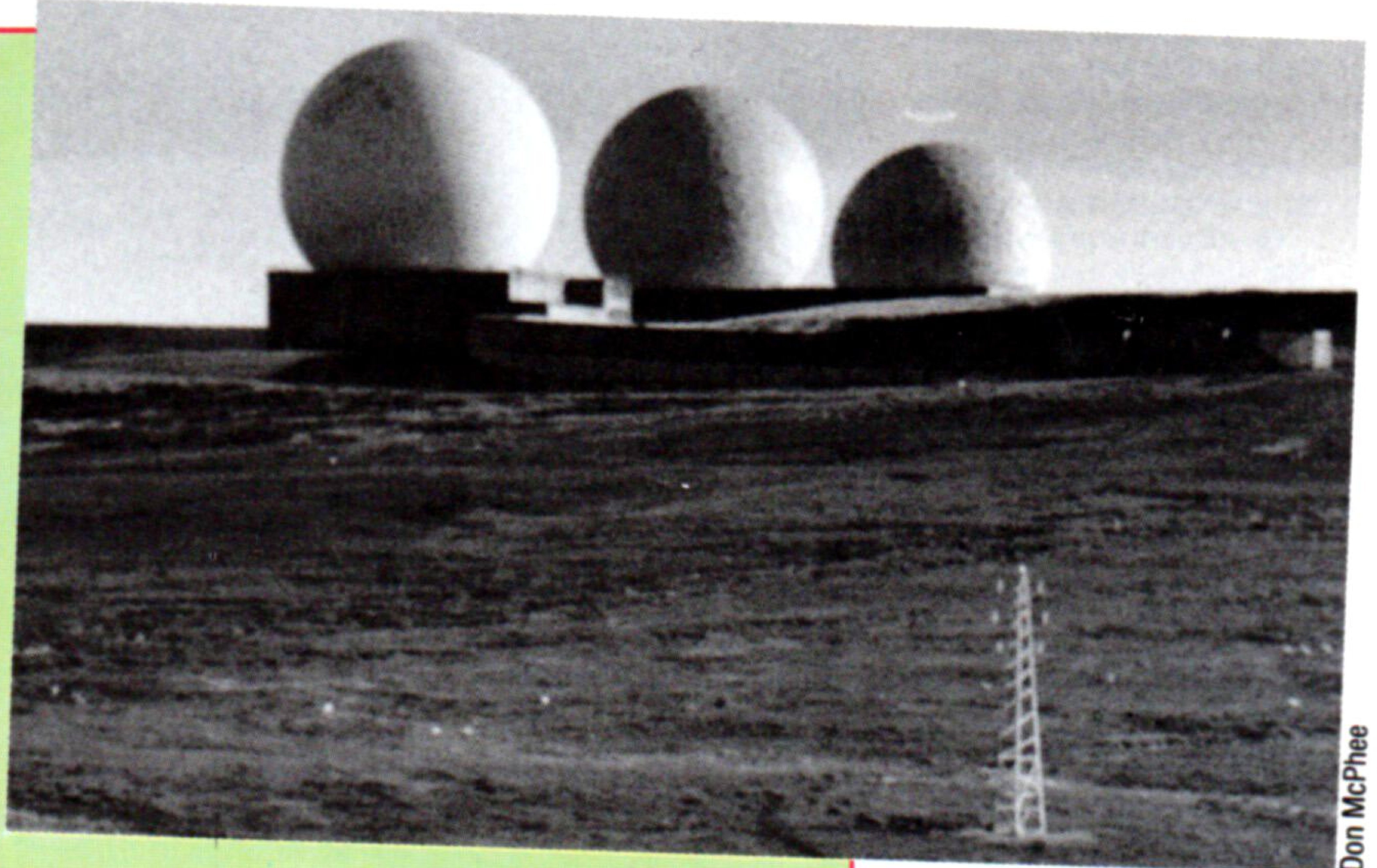
Las dos sentencias PRINT imprimirán los valores 8 y 10, respectivamente; el valor de FNy(2) y FNx(1) no es el mismo en ambos casos. Cambiando el orden de los términos también se altera el valor de la expresión, de modo que FNy(2)+FNx(1) no es lo mismo que FNx(1)+FNy(2). Para comprender la mayoría de los programas, hemos de "ejecutarlos" mentalmente utilizando un lápiz y un trozo de papel. Por el contrario, las sentencias matemáticas son independientes de la historia. Sabemos que las expresiones:

$(3+4)*(6-2)=(4+3)*(6-2)=7*4=28$

tienen todas el mismo valor, porque las "leyes" matemáticas nos dicen que 3+4 es lo mismo que 4+3, que 7 se puede sustituir legalmente por cualquiera de las dos expresiones, y que el valor de una expresión es el valor de sus componentes. Los programas escritos en lenguajes para ordenador convencionales no poseen ninguna de estas propiedades.

Si se pudiera lograr que los programas poseyeran estas propiedades matemáticas, se obtendrían varios beneficios inmensos y tendríamos una posible solución para la crisis.

- Los programas serían legibles. El texto del programa expresaría su significado sin ninguna necesidad de rastrear la historia de su ejecución.
- Los programas se podrían demostrar (como los teoremas) como correctos para todas las entradas posibles, en lugar de comprobarse meramente para un puñado de las posibles entradas. Esta prueba sería automatizada y la llevaría a cabo el ordenador.
- Programas ineficaces pero correctos se podrían transformar en otros más eficaces utilizando leyes como aquellas que rigen para las expresiones matemáticas. Este proceso podría automatizarse



Don McPhee

A la espera de una solución
Producir software para sistemas basados en grandes ordenadores, tales como el sistema de alerta rápida de Fylingdale, en North Yorkshire (Gran Bretaña), constituye un problema. Intentar conectar y racionalizar el trabajo de muchos programadores en un único gran trozo de código utilizando las técnicas de programación tradicionales es muy difícil, y las ulteriores modificaciones pueden conducir a una situación de difícil salida. Se espera que los lenguajes funcionales proporcionen la solución definitiva a este problema

En un lugar de Escocia...
El lenguaje funcional HOPE lo desarrollaron R. M. Burstall y D. B. MacQueen, del Departamento de Informática de la Universidad de Edimburgo (Escocia), con la colaboración de D. T. Sannella, de Bell Labs, de Estados Unidos. El nombre del lenguaje hace referencia a Hope Square, en Edimburgo, donde está situado el Departamento de Informática



Por las mismas líneas

Uno de los grandes atractivos del estilo de programación funcional es que se presta a la ejecución en paralelo. Ello se debe a que las partes de un programa funcional son independientes entre sí, gracias a la falta de variables.

En lenguajes convencionales, el empleo de variables compartidas por distintas partes del programa supone enormes problemas para un ordenador en paralelo. Supongamos que hemos inventado una forma de BASIC en paralelo y que estos dos programas se están ejecutando simultáneamente:

```
10   A=0              10   B=0
20   FOR X=1 TO K     .
.                     .
.                     .
100  NEXT X           .
2000 A=B+56           2000 PRINT A
.                     .
.                     .
```

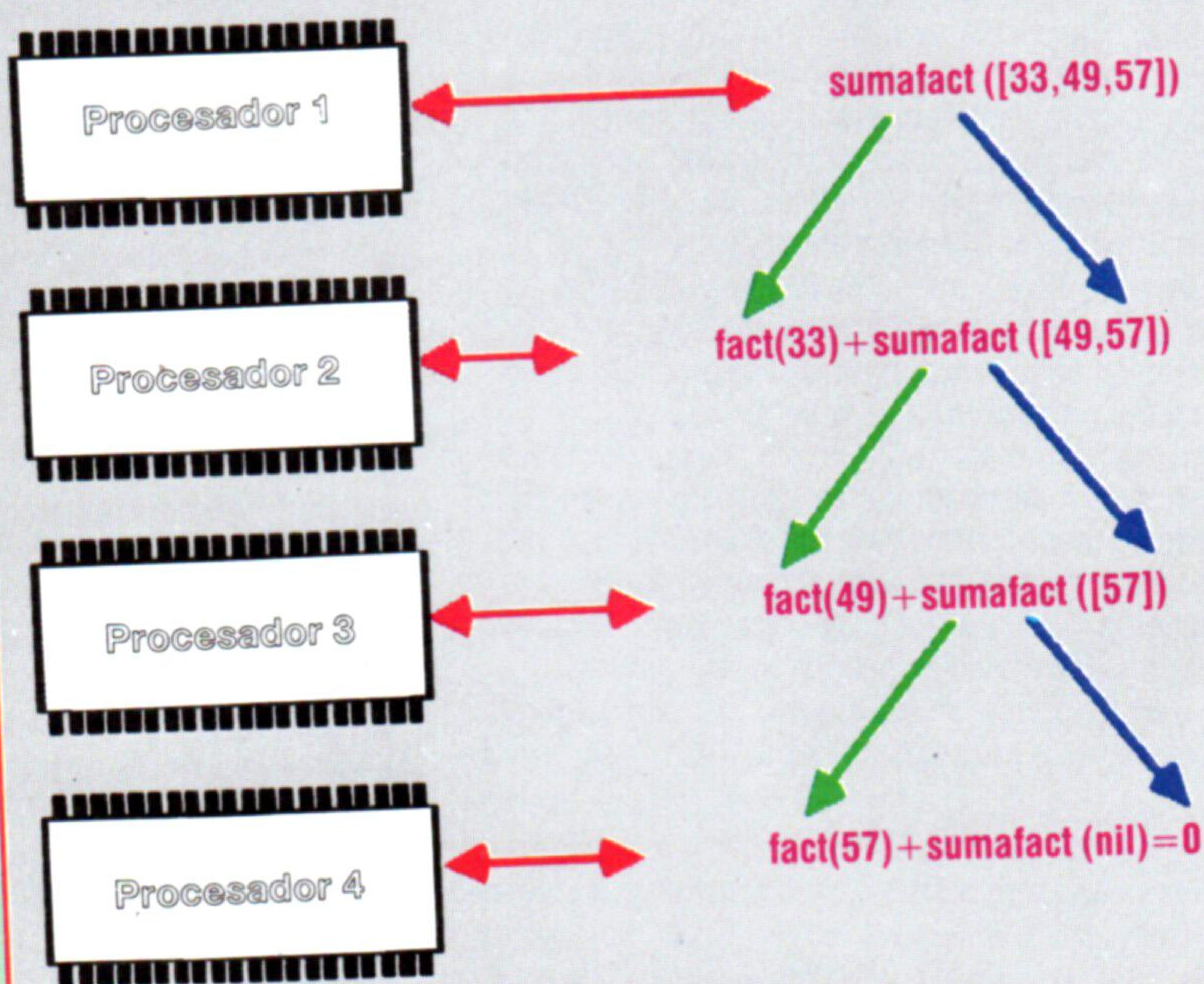
El valor que se imprima para A dependerá de que el primer programa haya llegado o no para entonces a su línea 2000. Si no ha llegado, A todavía es 0, de lo contrario habrá cambiado. Pero el tiempo de ejecución del primer programa varía con K y, por tanto, no podemos estar seguros del efecto que tendrá el programa. En un lenguaje como el HOPE, sin asignación a variables, no se presenta este problema. Tomemos a modo de ejemplo un programa para calcular la suma de los factoriales de una lista de números. Ya hemos visto la función factorial:

```
    dec fact:num—> num;
--- fact(0)        <=1;
--- fact(succ(n))<=(succ(n)*fact(n));
```

Ahora podemos escribir sumafact utilizando fact:

```
    dec sumafact:list(num)—>num;
--- sumafact(nil)<=0;
--- sumafact(x::y)<=fact(x)+sumafact(y);
```

Aquí decimos que la sumafact de una lista vacía es cero, de lo contrario se forma mediante la adición del factorial del primer elemento a la sumafact del resto. Los dos términos del lado derecho de la segunda ecuación son independientes entre sí y, por lo tanto, se pueden evaluar en paralelo.



En cada etapa se puede traer un nuevo procesador para calcular el término fact, y el primer procesador suma todos los resultados parciales cuando la recursión se "desenrolla" hasta llegar a 0. El ordenador en paralelo Alice que se está desarrollando en Imperial College ejecuta el HOPE de esta manera, utilizando un anillo de Transputers. Cualquier procesador de anillo que esté ocioso puede asumir uno de los cálculos parciales, que se ponen en "oferta" haciendo circular sus descripciones alrededor del anillo

Tamaño económico

Desarrollaremos una de las técnicas de compresión de textos mediante un programa de propósito general

El método de compresión que empleamos en nuestro programa es una implementación reducida del algoritmo de compresión de cuatro bits que describimos en el capítulo anterior. Trabaja traduciendo el texto a bloques de cuatro bits, cada uno de los cuales es o un carácter utilizado comúnmente, o bien un código que proporciona información relativa a los cuatro bits siguientes. El significado de los distintos códigos de cuatro bits se refleja en el diagrama *Valores de nibbles*.

Tal como está, el programa sólo comprimirá texto compuesto en letras mayúsculas, espacios, comas y puntos. Asimismo, soportará distintivos, y 16 de éstos se han incluido en el programa para ilustrar esta facilidad.

El programa exige que el usuario especifique la dirección de la serie de entrada y la dirección en donde se ha de colocar la serie de salida. La serie se almacena como un byte de cuenta, seguido por los bytes de la serie. La longitud de la serie se limita a 255 bytes.

El programa es relativamente sencillo de modificar, de modo que los programadores de lenguaje máquina no tendrán problemas para convertirlo en una implementación completa. La modificación más eficaz sería incluir una facilidad para hacer referencia a una tabla de distintivos de 255 elementos. La misma le permitiría incluir tanto caracteres en mayúsculas como en minúsculas, así como más distintivos para palabras comunes. Esta facilidad se podría introducir fácilmente destinando un valor nibble de tres para significar "tratar a los ocho bits siguientes como un desplazamiento en una tabla de 255 elementos". Se insertaría antes de la comprobación de distintivos de ocho bits y esencialmente sería igual que la rutina existente, exceptuando la tabla que direcciona.

Si decide introducir esta modificación, debe reacomodar las tablas de modo que los valores de caracteres de cuatro bits representen a las letras *minúsculas* más comunes (más caracteres de espacio y salto de línea). Las siguientes letras minúsculas más comunes van en la tabla de ocho bits, y las letras menos comunes (más mayúsculas y distintivos) se colocan en la tabla nueva.

La dirección ORG del programa se puede cambiar para adecuarla a diferentes micros. El programa se ha de ensamblar, y el código objeto almacenar en cinta o disco como preparación para el próximo capítulo, en el que ofreceremos el programa en BASIC para activar la utilidad.

Códigos de compresión

0 0 0 0 — Tratar al siguiente nibble como un valor de carácter

0 0 0 1 — Tratar al siguiente nibble como un valor de distintivo

0 0 1 0 — Fin del texto

0 0 1 1 … 1 1 1 1 — 13 caracteres más comunes

Valores de nibbles
Nuestro programa codifica el texto en secuencias de cuatro bits (nibbles), cuyo significado se refleja en el diagrama. Observe que sólo se utilizan como "señales" los valores del 0 al 2. El programador, sin embargo, podría incluir el valor 3 como una señal de que los dos nibbles siguientes se han de tomar para indicar un desplazamiento en una tabla de 255 elementos, permitiendo, en consecuencia, la inclusión de caracteres en minúsculas y más distintivos

Programa compresor de textos

Nuestro programa de compresión de textos se ofrece en dos partes. Aquí presentamos un listado ensamblador para máquinas Z80, y en el próximo capítulo proporcionaremos una versión 6502. Asimismo, para ambas implementaciones se proporcionarán cargadores de BASIC, para aquellos lectores que no posean ensambladores

```
        org     30000

        jr      start                   ;saltar el espacio de datos

string: dw      0       ;cargar con dirección serie de entrada
output: dw      0       ;entrar dirección para serie de salida
status: db      0       ;retorno estado, señales 0 OK
mask:   db      0       ;
len:    db      0                       ;espacio de almacenamiento para longitud entrada

start:  ld      hl,(string)             ;tomar dirección de la serie a comprimir
        ld      a,(hl)                  ;tomar longitud serie de entrada
        ld      (len),a                 ;almacenarla
        inc     hl                      ;apuntar al comienzo de la serie
        ld      de,(output)             ;apuntar al espacio de salida
        push    de                      ;guardar comienzo de salida
        inc     de                      ;dejar 1 espacio para cuenta
        ld      (output),de             ;guardar dirección de salida hasta que
                                        ;tengamos algo para almacenar
        ld      a,255
        ld      (mask),a                ;tratar primero nibble izquierdo

nchar:  ld      a,(hl)                  ;tomar byte con el que trabajar
        call    check                   ;asegurarse de que esté en la gama
        jr      nz,badchar              ;jr si carácter no permitido
        call    token                   ;comprobar si comienzo de un distintivo
        jr      z,gottoken              ;hallado distintivo, procesarlo
        call    fourbit                 ;comprobar caracter de 4 bits
        jr      z,got4bit               ;el carácter es de 4 bits, saltar
        call    eightbit                ;tomar valor carácter 8 bits
        push    af                      ;no hay necesidad de comprobar-guardar valor
        ld      a,0                     ;mientras indicador de 'car de 8 bits'
```

```
          call   writenib      ;..salida
          pop    af            ;restaurar 2ºnibble de val.8 bit
got4bit:  call   writenib      ;salida valor del 2ºnibble de 8 bits,o
                               ;del 1er nibble de 4 bits
rejoin:   inc    hl
          ld     a,(1en)       ;volver a tomar longitud actual
          dec    a             ;menos uno para car. recién procesado
          ld     (len),a       ;volver a poner nueva longitud
          and    a
          jr     naz,nchar     ;jr para proc. car. siguiente si existe
          ld     (status),a    ;señal terminado ok, a será=0
          ld     a,2           ;2=fin del texto comprimido
          call   writenib      ;escribirlo
          ex     de,hl         ;dirección de salida final en hl
          pop    de            ;tomar dirección serie de salida
          ld     a,(mask)      ;tomar valor máscara
          and    a
          jr     z,nodec       ;si máscara cero entonces valor final ok
          dec    hl            ;ignorar byte final, es nulo
nodec:    and    a             ;limpiar acarreo para resta
          sbc    hl,de         ;tomar longitud
          ld     a,l           ;tomar longitud - no más de 255!
          ld     (de),a        ;almacenar cuenta al comienzo de salida
          ret                  ;retornar

badchar:  pop    de            ;limpiar pila
          ld     a,255
          ld     (status),a    ;fallo de señal
          ret
gottoken:                      ;
          push   af            ;guardar nibble distintivo
          ld     a,1           ;viene distintivo señal
          call   writenib
          pop    af            ;restaurar distintivo
          call   writenib      ;salida distintivo
          jr     rejoin

          ;subrutina para comprobar comienzo de nuevo distintivo
token:    push   hl            ;guardar dirección de entrada
          ex     de,hl         ;poner dirección de entrada en de
          ld     hl,tktable    ;que hl apunte a tabla de distintivos
          ld     a,(len)       ;poner longitud restante en a
          ld     c,a           ;y transferirla a c
          ld     b,0fh         ;inicializar número distintivo
tok1:     ld     a,(hl)        ;tomar longitud de distintivo actual
          and    a
          jr     z,notfound    ;si longitud=0 luego findelatabla
          push   de            ;guardar estos punteros por si
          push   hl            ;son necesarios
          inc    hl            ;apuntar a primer car.de distintivo
          push   bc            ;guardar número de distintivo
          ld     b,a           ;cuenta en b
tok2:     ld     a,(de)        ;tomar carácter de entrada
          cp     (hl)          ;comparar con carácter distintivo
          jr     nz,nxttoken   ;si no concuerda olvidar este distintivo
          inc    hl            ;apuntar al siguiente carácter
          inc    de
          dec    b             ;reducir cuenta caracteres restantes
          jr     nz,chkchars   ;si b no es cero entonces saltar
          ld     a,c
          ld     (len),a       ;almacenarlo
          pop    bc            ;hallado distintivo-restaurar n.º distintivo
          pop    hl            ;limpiar pila
          pop    hl            ;
          pop    hl            ;
          ex     de,hl         ;poner nueva dirección de entrada en hl
          dec    hl            ;y dec para mantener ok el bucle principal
          xor    a             ;establecer flag a cero para indicar éxito
          ld     a,b           ;poner número distintivo en a
          ret
chkchars:
          dec    c             ;reducir cuenta caracteres restantes
          jr     nz,tok2       ;si aún hay caracteres para comprobar volver
nxttoken:
          pop    bc            ;restaurar caracteres restantes+n.º distintivo
          dec    b             ;siguiente número de distintivo
          pop    hl            ;volver a poner puntero en número distintivo
                               actual
          ld     a,(hl)
          ld     e,a           ;longitud en de
          ld     d,0
          inc    hl            ;saltar cuenta
          add    hl,de         ;ahora hl apunta a cuenta siguientes
                               distintivos
          pop    de            ;restaurar dirección de entrada
          jr     tokl          ;saltar para comprobar siguiente distintivo

notfound:
          pop    hl            ;restaurar dirección de entrada original
          ld     a,(hl)        ;restaurar carácter
          or     a             ;restaurar flag a cero para indicar fracaso
          ret

          ;subrutina para comprobar caracteres de 4 bits

fourbit:  push   hl            ;guardar dirección de entrada
          ld     hl,tab4bit    ;apuntar hl a tabla de caracteres
          call   tabscan       ;explorar tabla para valor
          pop    hl            ;restaurar dirección de entrada
          ret                  ;volver

          ;subrutina para comprobar caracteres de 8 bits
eightbit:
          push   hl            ;guardar dirección de entrada
          ld     hl,tab8bit    ;apuntar hacia tabla caracteres de 8 bits
          call   tabscan       ;explorar tabla
          pop    hl            ;restaurar dirección
          ret                  ;return

          ;rutina de exploración de tabla de propósito general

tabscan:  ld     b,0fh         ;16 caracteres en cada tabla
                               ;(uno se numera 0)
tabsc2:   cp     (hl)          ;comprobar carácter
          jr     z,tabsc3      ;retornar con 0 establecido si concordancia
          inc    hl            ;comprobar siguiente elemento de la tabla
          djnz   tabsc2        ;bucle hasta que ningún carácter en la tabla
          or     a             ;restablecer bandera a 0 para indicar ninguna
                               pareja
                               ;a contiene código car <>0
          ret                  ;volver
tabsc3:   ld     a,b           ;poner nibble en a
          ret

          ;rutina para comprobar caracteres válidos
          ;sólo se permiten espacio , . ; : y letras mayúsculas
check:    cp     ''            comprobar espacio primero
          ret    z
          cp     ','           ;y coma
          ret    z
          cp     '.'           ;por último punto
          ret    z
          cp     'A'           ;asumir abecedario contiguo
          jr     c,nogood      ;saltar si 'menor que' A
          cp     'Z'+1
          jr     nc,nogood
          ld     c,a
          xor    a             ;establecer cero para indicar carácter ok
          ld     a,c
          ret
nogood:   ld     a,255
          and    a             ;restablecer cero para indicar fracaso
          ret

writenib:
          ld     c,a           ;guardar nibble
          ld     a,(mask)      ;
          ld     de,(output)   ;tomar dirección de byte para escribir
          and    a             ;comprobar valor máscara
          jr     nz,left       ;saltar si el nibble requiere desplazamiento
          ld     a,(de)        ;tomar nibble antiguo
          or     c             ;insertar nibble nuevo
          ld     (de),a        ;y volver a almacenarlo
          inc    de            ;apuntar al siguiente byte
          ld     (output),de   ;guardar dirección
          ld     a,255
          ld     (mask),a      ;apuntar nibble izquierdo próxima vez
          ret
left:     ld     a,c           ;poner valor en a
          sla    a             ;desplazarlo
          sla    a
```



```
        sla     a
        sla     a
        ld      (de),a          ;y almacenarlo
        xor     a
        ld      (mask),a        ;señalar otro nibble la próxima vez
        ret

;programa de ampliación de texto 4 bits para Z80, independiente de la máquina

expand: ld      hl,(string)     ;string contiene la dirección de la serie a
                                ;expandir
        inc     hl              ;apuntar hl al comienzo de la serie
        ld      (string),hl     ;volver a poner comienzo nueva serie
        ld      de,(output)     ;apuntar al comienzo espacio de salida
        push    de              ;guardar para final
        ld      a,255
        ld      (mask),a        ;tratar primer nibble izquierdo

nextnib: call   getnib          ;tomar siguiente nibble de entrada
        cp      2               ;es este el final
        jp      z,fin           ;de ser así saltar para terminar
        jp      nc,exp4bit      ;si nibble>2 procesar 4 bits
        and     a
        jp      z,exp8bit       ;si nibble=1 procesar car.8 bits
        call    getnib          ;tomar nibble para ampliar a distintivo
        call    tokscan         ;hallar el que deseamos
tokloop: ld     b,a             ;cuenta en b
tklp1:  ld      a,(hl)          ;tomar carácter distintivo
        inc     hl              ;apuntar al siguiente carácter
        call    outchar         ;salida a serie nueva
        djnz    tklp1           ;bucle para todos los cars. distintivo
        jr      nextnib         ;saltar hacia atrás para siguiente nibble
                                entrada
tokscan: ld     hl,tktable
        ld      b,a             ;inicializar b
        ld      a,0fh
        sub     b
        ld      b,a             ;cuenta en b
        inc     b
toksc1: ld      a,(hl)          ;tomar longitud de este distintivo
        inc     hl              ;apuntar al primer carácter de distintivo
        dec     b               ;reducir cuenta
        ret     z               ;retornar si es el que queremos
        ld      e,a             ;saltar este distintivo
        ld      d,0
        add     hl,de
        jr      toksc1          ;saltar hacia atrás

fin:    ld      hl,(output)     ;poner fin de serie de salida en hl
        pop     de              ;comienzo de la serie de salida
        and     a               ;limpiar acarreo
        sbc     hl,de           ;longitud en hl
        ld      a,l             ;la longitud no debe ser > 255
        ld      (de),a          ;poner cuenta en lugar correcto
        ret                     ;retornar al sistema

exp8bit: call   getnib          ;tomar siguiente nibble para car.8 bits
        ld      hl,tab8bit      ;que hl apunte a la tabla correcta
        call    index           ;tomar el carácter derecho
        call    outchar         ;ponerlo en la serie de salida
        jr      nextnib         ;volver por más

exp4bit: ld     hl,tab4bit      ;apuntar a tabla 4 bits
        call    index           ;hallar carácter
        call    outchar         ;hacerlo salir
        jr      nextnib         ;volver a pasar

index:  ld      e,a             ;inicializar e
        ld      a,0fh
        sub     e               ;poner desplazamiento en a
        ld      e,a             ;poner en e
        ld      d,0
        add     hl,de           ;índice en tabla
        ld      a,(hl)          ;tomar carácter
        ret

outchar: ld     de,(output)     ;tomar última dirección producida para
        inc     de              ;apuntar a nueva dirección
        ld      (de),a          ;almacenar carácter
        ld      (output),de     ;volver a poner nueva dirección
        ret

getnib: ld      a,(mask)        ;tomar valor máscara
        ld      hl,(string)     ;tomar dirección de byte de entrada
        and     a
        ld      a,(hl)          ;tomar byte
        jr      nz,shfta        ;saltar si requerido nibble izquierdo
        and     0fh             ;enmascarar nibble derecho
        inc     hl              ;apuntar a nuevo byte
        ld      (string),hl     ;almacenar para próxima vez
        ld      c,a
        ld      a,255           ;indica nibble izquierdo próxima vez
        ld      (mask),a
        ld      a,c
        ret

shfta:  sra     a               ;desplazar nibble derecho por arriba
        sra     a
        sra     a
        sra     a
        and     0fh             ;desenmascararlo
        ld      c,a
        xor     a               ;indica nibble derecho próxima vez
        ld      (mask),a
        ld      a,c
        ret
        ;tabla de valores

        ;tabla de cuatro bits

tab4bit: db     ' '             ;carácter espacio
        db      'E'
        db      'T'
        db      'A'
        db      'O'
        db      'N'
        db      'R'
        db      'I'
        db      'S'
        db      'H'
        db      'D'
        db      'L'
        db      'F'
        db      0               ;valores ficticios para llenar tabla
        db      0
        db      0

        ;caracteres de 8 bits

tab8bit: db     'C'
        db      'M'
        db      'U'
        db      'G'
        db      'Y'
        db      'P'
        db      'W'
        db      'B'
        db      'V'
        db      'K'
        db      'X'
        db      'J'
        db      'Q'
        db      'Z'
        db      ','
        db      '!'

        ;tabla de distintivos
tktable: db     3,'THE'
        db      4,'THIS'
        db      4,'THAT'
        db      2,'IF'
        db      3,'YOU'
        db      2,'ME'
        db      3,'WAS'
        db      2,'HE'
        db      3,'SHE'
        db      4,'THEY'
        db      2,'OF'
        db      2,'IT'
        db      2,'IS'
        db      3,'FOR'
        db      2,'ON'
        db      2,'TO'
        db      0               ;marca de fin de tabla
        end
```

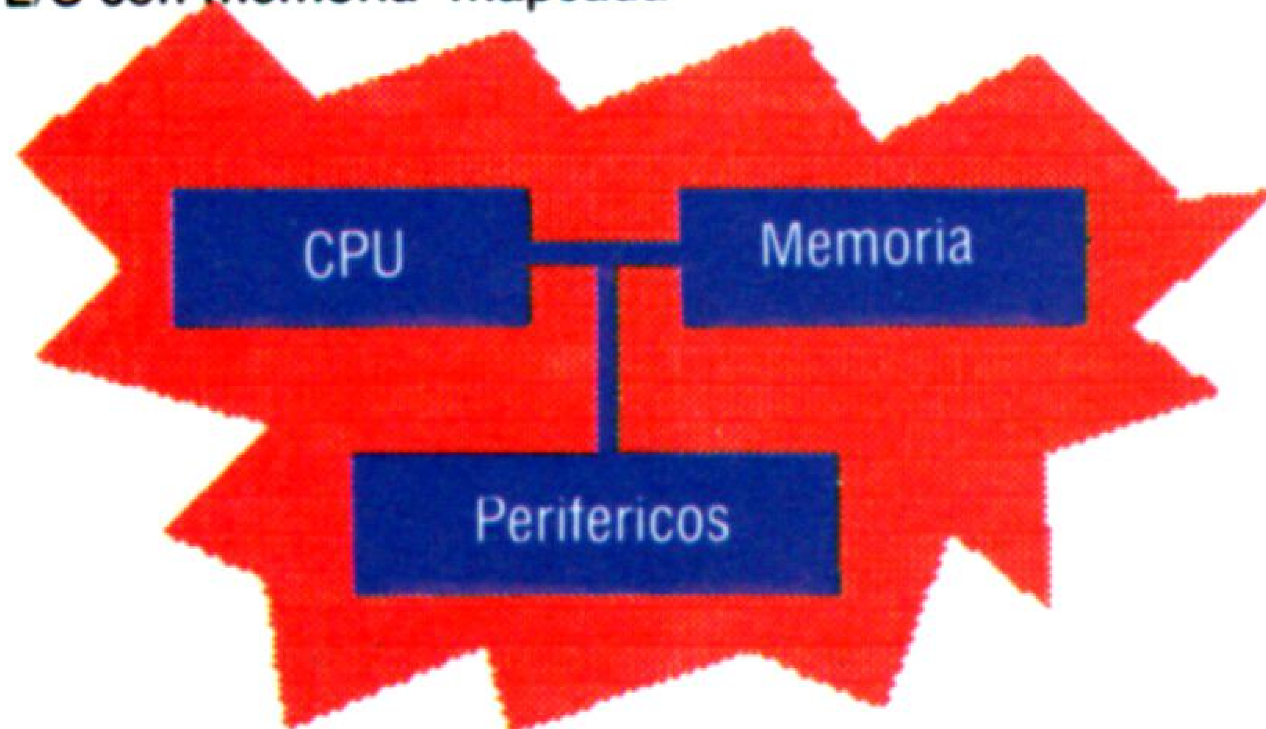
Bit... bit... bit

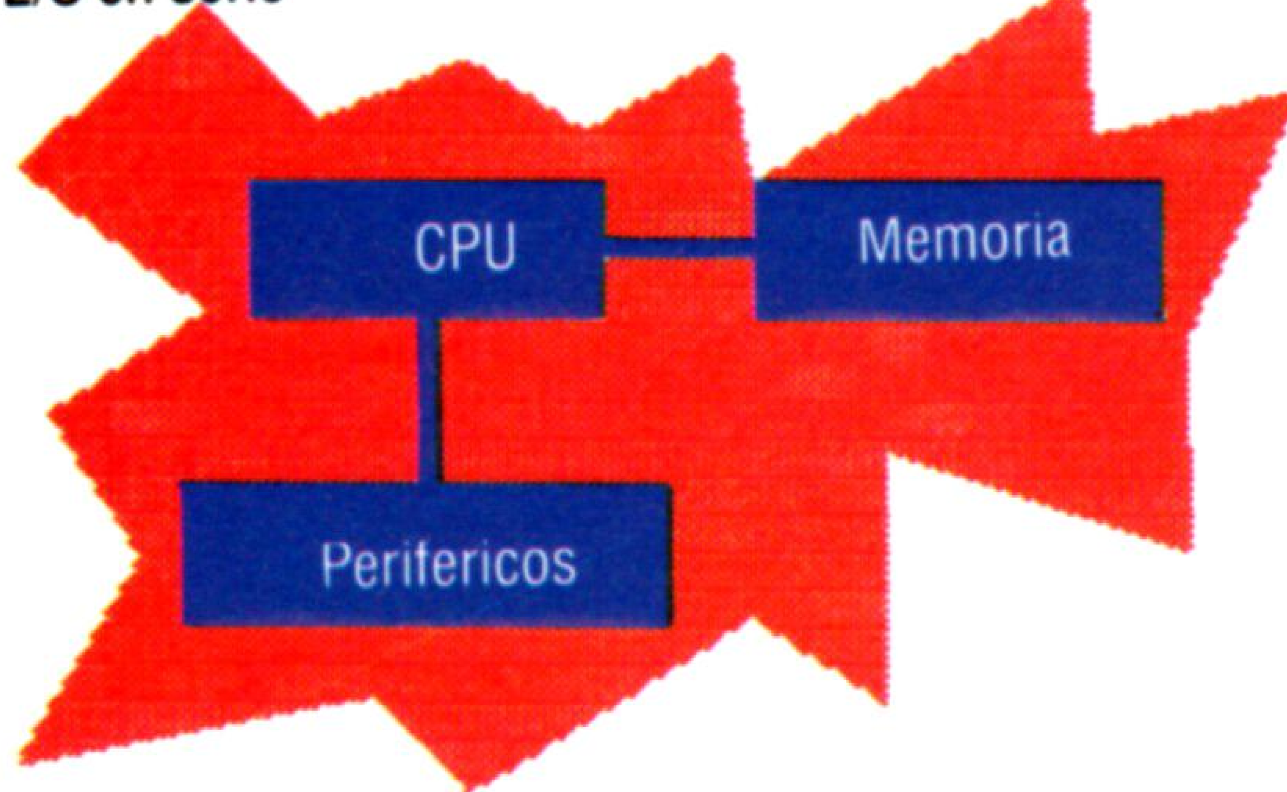
¿Cuál es la información necesaria para que la CPU saque el mayor partido del mapa de memoria en las comunicaciones de E/S? Veámoslo

Hay dos métodos básicos para seleccionar y comu nicarse con dispositivos de E/S:

- *Memoria mapeada*: Este método conecta el bus central de comunicaciones (al que están conectadas la CPU y la memoria) con los dispositivos de E/S. Mediante la adecuada combinación electrónica, los periféricos son sencillamente agregados a este bus. Lo que significa que un periférico se convierta en memoria mapeada, dado que el procesador puede leer y escribir ahora en el dispositivo como si se tratara de una memoria normal. Este procedimiento presenta la desventaja de ralentizar la velocidad en el bus, pues el periférico entra en competencia para obtener sus servicios.

- *E/S aisladas*: En este método se dispone de un bus especial para los dispositivos de E/S, con lo que se incrementa la velocidad de transferencia de los datos respecto al procedimiento anterior. El inconveniente está en que hay que conocer las nuevas instrucciones de E/S que rigen los dispositivos.





Caroline Clayton

Líneas de los buses
Este dibujo muestra la diferencia entre una E/S en serie, que emplea un bus de periférico especial para intercambiar información, y una E/S de memoria mapeada, donde los periféricos toman los datos del mismo bus con el que se accede a la memoria. En este último caso ciertas direcciones habrán de ser reservadas para los periféricos, a ellas puede acceder el ordenador de igual modo que a la RAM o a la ROM

Llegados a este punto no puede sorprendernos saber que el 68000 opta por el método de memoria mapeada. Ya estudiamos en otro lugar la estructura del microordenador. Vimos entonces que algunos dispositivos se conectaban al mismo bus que la memoria y que la CPU, lo que equivalía a decir unas E/S mapeadas.

Veamos ahora los tres tipos principales de información que el procesador exige, y que es mapeado en la memoria.

- *Estado*: Dado que los periféricos tardarán algún tiempo para realizar la operación deseada (imprimir un carácter, leer el teclado, p. ej.), es necesaria una información de su estado para impedir que el periférico reciba órdenes antes de haber acabado su última operación. También es necesaria otra información no directamente referida a la operación pero que puede interesar, supongamos, al modo de operación o a cualquier condición de fallo como quedarse sin papel en la impresora.
- *Control*: Necesitamos un registro que nos permita configurar o activar el periférico que estamos usando (p. ej., ordenar a una unidad de disco que lea un bloque de datos, o configurar un canal de comunicación a una velocidad específica de transmisión).
- *Datos*: Necesitamos, por último, poder leer o escribir datos en el periférico y retener la información allí hasta que se concluya la operación y quede listo para recibir nueva información.

Chips de E/S

En el campo de las señales en el bus principal del ordenador, la operación detallada de los chips de interface puede resultar bastante complicada. Para evitar esta complicación (desde un punto de vista de la programación) se utilizan chips de interface de usos específicos. Motorola proporciona dos chips, uno para el control de dispositivos en serie (ACIA) y otro para el control en paralelo (PIA).

Los dispositivos en serie se conectarán al chip mediante dos hilos solamente para envío y dos para recepción de información; los bits que constituyen los bytes de datos llegan en serie uno detrás de otro. En el caso de dispositivos en paralelo los bits de cada byte van juntos y al mismo tiempo. El periférico en sí se conecta con el chip a través de un número de un byte (o incluso de una palabra) de líneas.

Una de las ventajas del empleo de estos chips de interface está en que el programa cobra así un alto grado de flexibilidad en el establecimiento de detalles relativos a la configuración del hardware. No obstante, esto significa que el número de bits en la palabra de control, especialmente, puede ser bastante grande. Por ejemplo, una asignación de bit de control ACIA es la siguiente:

Bits 0 y 1	frecuencia de división del reloj e inicialización (velocidad bit serial)
Bits 2 al 4	formato de carácter en serie (del tipo paridad o no, número de bits de stop)
Bits 5 y 6	bits de control del transmisor (como la activación de interrupciones)
Bit 7	bit de control del receptor se pone a uno para activar interrupciones



El orden en que se establecen estos bits debe primeramente asegurar que tenga lugar una comunicación aceptable entre el ordenador y el periférico. Si esto falla se pueden obtener datos sin sentido en el dispositivo receptor.

Se precisan dos instrucciones para una E/S en serie concreta. Una de ellas configura el ACIA para el periférico serial. Por ejemplo:

```
MOVE.B   #$3,ACIACON   inicializa el hardware
MOVE.B   #$15,ACIACON  configura el hardware
```

En este caso, ACIACON ha de estar previamente definido.

Tendremos igualmente bits de estado que se leerán para una correcta transferencia de datos. Por ejemplo:

Bit 0	registro receptor datos preparado (un nuevo carácter está disponible)
Bit 1	registro transmisión vacío (se puede enviar un nuevo carácter)
Bits 2 y 3	señales control modem (CTS y DCD)
Bits 4 a 6	indicaciones de error sobre los datos recibidos (p.e., error de paridad)
Bit 7	petición de interrupción

Las interrupciones serán estudiadas en el próximo capítulo, pero los modems caen fuera del ámbito de esta serie.

Dado que sólo leemos un registro de estado y escribimos en un registro de control, podríamos usar sólo una dirección para lograr el mapeo de la memoria. Esto es igualmente aplicable al registro de datos, ya que sólo leemos de un registro receptor de datos y escribimos en un registro transmisor. Como ejemplo supongamos que queremos entrar un carácter,

```
INCH BTST   #0,ACIASTAT      ¿hay carácter
                             disponible?
     BEQ    INCH             si no es así, seguir
                             probando
     MOVE.B ACIADATA,D0      coloca el carácter
                             como parámetro
     RTS                     retorno subrutina
```

En este caso, esperamos hasta que el bit 0 se active en el registro de control antes de leer un registro de datos. ACIASTAT es igual que ACIACON en este ejemplo.

De modo similar podemos escribir una subrutina para salida de caracteres:

```
OUTCH BTST   #1ACIASTAT    espera hasta
                           que el reg. de
                           datos esté
                           vacío
      BEQ    OUTCH         si no es así,
                           bifurcar y
                           volver a
                           comparar
      MOVE.B D0,ACIADATA   salida carácter

      RTS
```

Si es necesario repetir cualquier carácter recibido llamaríamos simplemente a una subrutina después de la otra:

```
ECHO  JSR    INCH     lee un carácter
      JSR    OUTCH    salida carácter
      BRA    ECHO     iteración
```

Es preciso subrayar aquí que la ejecución de una E/S de esta manera es poco eficaz dado que el ordenador trabaja a la velocidad de la transmisión de los caracteres. Aun así no es útil para ilustrar el mecanismo básico de una entrada/salida programada. En el próximo capítulo veremos cómo contribuyen las interrupciones a solventar este problema y cómo se ejecutan las transferencias de datos en E/S en paralelo.

Resumen de instrucciones del 68000

Antes de pasar a examinar las interrupciones en el 68000, vamos a proporcionar un resumen de todas las instrucciones del 68000 que hemos estudiado. La primera lista es el grupo que sirve para copiar datos y gira en torno a la instrucción MOVE:

Instrucción	Operación
MOVE	Mueve la fuente al destino
MOVEM	Mueve los registros de dirección y datos a y de la memoria (útil en entradas y salidas de subrutinas)
MOVEA	Mueve el contenido de la dirección efectiva al registro de direcciones
LEA	Mueve la dirección fuente al registro de direcciones
MOVEQ	Mueve con rapidez pequeñas constantes al registro de datos
PEA	Coloca la dirección efectiva en la pila
SWAP	Intercambia palabras superiores e inferiores en un registro de datos
EXG	Intercambia palabras largas entre determinados registros
LINK/UNLK	Empleado para establecer parámetros para llamadas a subrutinas (especializadas)

Este conjunto de instrucciones proporciona algunas facilidades bastante amplias, pero hay que estar atento en el empleo de instrucciones tan exóticas como LINK/UNLK. Veamos ahora las instrucciones encargadas de la aritmética con números enteros:

Instrucción	Operación
ADD	Suma fuente a destino
ADDA	Suma registro direcciones fuente a destino
ADDI	Suma datos inmediatos a destino de datos alterable
ADDQ	Suma rápida inmediata para constantes pequeñas (del 1 al 8)
ADDX	Suma con arrastre
SUB(A/I/Q/X)	Resta fuente de destino en cualquiera de las variantes de direccionamiento admitidas por ADD
CMP(A/I)	Compara fuente con destino y establece códigos de condición
CMPM	Compara posiciones de memoria y punteros de posincremento

NEG(X)	Hace negativo el registro de datos de operando
EXT	Amplía el signo en el registro de datos de operando
MULU	Multiplica datos sin signo en el registro de datos por dirección efectiva y pone el resultado en el registro de datos
MULS	Como MULU, para datos con signo
DIVU(S)	Divide el registro destino de datos por el operando fuente. Cociente y resto quedan en el reg. destino
TST	Activa los códigos de condición según sea el operando
TAS	Comprueba y activa el bit más significativo del operando
CLR	Limpia la dirección efectiva

Un buen racimo de instrucciones, ¡pero es necesario tener cuidado al seleccionar la instrucción adecuada para los modos de direccionamiento del operando! Sigue ahora una lista de instrucciones relativas a la aritmética en BCD:

Instrucción	Operación
ABCD	Suma operandos BCD colocando el resultado en el destino
SBCD	Resta los operandos
NBCD	Pone negativo el operando BCD

Es de notar que no existe ninguna instrucción BCD para multiplicar. Sin embargo, las instrucciones lógicas que ahora siguen colman estas necesidades:

Instrucción	Operación
AND	Opera con AND el fuente y el destino, siendo uno de ellos al menos un registro de datos
ANDI	AND lógico con datos inmediatos como fuente
OR	Opera con OR de modo semejante a la AND anterior
ORI	OR lógico con datos inmediatos
EOR	OR exclusivo
EORI	OR exclusivo con datos inmediatos
NOT	Inversión lógica del operando

Es el turno de las instrucciones con comprobación y manipulación de bits:

GRUPO MANIPULADOR DE BITS

Instrucción	Operación
BTST	Comprueba el bit especificado en el operando fuente
BSET	Comprueba y activa el operando destino
BCLR	Comprueba y limpia
BCHG	Comprueba y cambia el operando

GRUPO PARA DESPLAZAMIENTO Y ROTACION

Instrucción	Operación
ASl	Desplazamiento aritmético a izq..
ASR	Desplazamiento aritmético a der.
LSL	Desplazamiento lógico a izq.
LSR	Desplazamiento lógico a der.
ROL	Rotación a izq. del operando, activando adecuadamente el arrastre
ROR	Rotación a der. del operando, activando el bit de arrastre

Por último revisaremos las instrucciones de subrutinas y control de programas:

INSTRUCCIONES DE CONTROL DE PROGRAMAS

Instrucción	Operación
BRA	Bifurca siempre (forma eficaz de bifurcación incondicional)
JMP	Salta siempre (útil cuando se desea hacer algún cálculo en la dirección del salto)
Bcc	Bifurcación condicionada según el código de condición 'cc' que se comprueba. Es decir:

Para operandos con signo:

GT	mayor que
LT	menor que
GE	mayor o igual que
LE	menor o igual que
VS	desbordamiento activado
VC	desbordamiento limpio

Para operandos sin signo:

EQ	igual a
NE	no igual a
MI	menos
PI	más
HI	superior a
LS	inferior o igual que
CS	arrastre activado
CC	arrastre limpio

DBcc	Contador de bucle en decremento y bifurcación según condición de 'cc', como antes. ¡Recordar que la instrucción tiene un sentido diferente!

INSTRUCCIONES DE CONTROL DE SUBRUTINAS

Instrucción	Operación
JSR	Salto a subrutina
BSR	Forma eficiente de JSR
RTS	Retorno de subrutina

Este resumen no incluye todas las instrucciones del 68000. Se ha omitido deliberadamente un grupo especial relativo al control del sistema que emplea instrucciones privilegiadas. Estas instrucciones son empleadas en general por los ingenieros de sistemas operativos, por lo que están fuera de la finalidad de este estudio. Las instrucciones se refieren a operaciones sobre el registro de estado y los punteros de la pila, y las instrucciones de "interrupciones software" llamadas *trampas*. Naturalmente, pueden verse en detalle en el manual del usuario del 68000


Editorial Delta, S.A.
mi COMPUTER




10117

9 788485 822836